Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____

КАФЕДРА _____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

_____

_____

_____

_____

_____

_____

Студент _____ _____ _____
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ _____
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

20 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой _____
(Индекс)
_____ _____
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине _____

Студент группы _____

_____
(Фамилия, имя, отчество)

Тема курсовой работы _____
_____
_____

Направленность КР (учебная, исследовательская, практическая, производственная, др.)
_____
Источник тематики (кафедра, предприятие, НИР) _____

График выполнения работы: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Задание _____
_____
_____
_____
_____

Оформление курсовой работы:

Расчетно-пояснительная записка на _____ листах формата А4.

_____
_____
_____
_____

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсовой работы _____ _____
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Курсовой проект по курсу "Технологии машинного обучения"

Горбатнко И.А. ИУ5-64

1) Поиск и выбор набора данных для построения моделей машинного обучения. Построение модели машинного обучения для решения задачи классификации и задачи регрессии на основе выбранного набора данных

В качестве набора данных возьмем базу данных наблюдаемых пациентов с возможным сердечно-сосудистым заболеванием. База состоит из 14 атрибутов:

- age - возраст пациента
- gender - пол пациента (0 или 1)
- chest_pain_type - тип боли в груди (значения от 0 до 3)
- blood_pressure - кровяное давление в состоянии покоя в мм.рт.ст.
- cholestoral - количество холестерина в мг/дл
- sugar - количество сахара в крови (1, если >120мг/дл, 0, если <=120мг/дл)
- ECG - электрокардиографические результаты в состоянии покоя (значения от 0 до 2)
- max_heart_rate - максимальное зафиксированное значение пульса
- stenocardia - наличие стенокардии или ее отсутствие после физической нагрузки (0 или 1)
- ST_depression - депрессия ST, вызванная физической нагрузкой относительно покоя
- slope - наклон пикового значения ST при нагрузке (от 0 до 2)
- vessels - количество крупных сосудов, показанных на флюороскопии (от 0 до 3)
- thal - 3 = нормальный; 6 = исправленный дефект; 7 = обратимый дефект
- target - наличие или отсутствие сердечно-сосудистого заболевания у пациента (1 или 0)

Конечной целью (target) является значение 0 или 1 (соответственно отсутствие сердечно-сосудистого заболевания или его наличие). Будем решать задачу классификации и задачу регрессии. В качестве целевого признака для решения задачи классификации будем использовать "target". "target" принимает значения только 0 или 1, значит это задача бинарной классификации. В качестве целевого признака для решения задачи регресии будем использовать "max_heart_rate". Датасет состоит из одного файла "Heart_Desease.csv", содержащий 303 строки.

Импортируем библиотеки:

In [1]:
```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classi
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_s
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, Lin
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, exp
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRe
from gmdhpy import gmdh
%matplotlib inline
sns.set(style="ticks")
```

Отрисовка ROC-кривой:

In [2]:
```
def draw_roc_curve(y_true, y_score, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

Поскольку у нас изначльно один файл датасета, напишем функцию split, деляющую csv-файл на куски и разделим наш датасет Heart_Desease.csv из 303 строк на два датасета: Heart_Desease_Train.csv из 212 строк - обучающая выборка, и Heart_Desease_Test.csv из 91 строки - тестовая выборка.

In [3]:
```
def split(filehandler, delimiter=',', row_limit=212,
          output_name_template='Heart_Desease%s.csv', output_path='.', keep
    import csv
    reader = csv.reader(filehandler, delimiter=delimiter)
    current_piece = 1
    current_out_path = os.path.join(
        output_path,
        output_name_template % current_piece
    )
    current_out_writer = csv.writer(open(current_out_path, 'w'), delimiter=
    current_limit = row_limit
    if keep_headers:
        headers = next(reader)
        current_out_writer.writerow(headers)
    for i, row in enumerate(reader):
        if i + 1 > current_limit:
            current_piece += 1
            current_limit = row_limit * current_piece
            current_out_path = os.path.join(
                output_path,
                output_name_template % current_piece
            )
            current_out_writer = csv.writer(open(current_out_path, 'w'), de
            if keep_headers:
                current_out_writer.writerow(headers)
        current_out_writer.writerow(row)
```

In [4]:
```
split(open('Heart_Desease.csv', 'r'));
```

In [5]:
```
os.rename('Heart_Desease1.csv', 'Heart_Desease_Train.csv')
os.rename('Heart_Desease2.csv', 'Heart_Desease_Test.csv')
```

Теперь зададим наши обучающую и тестовую выборки:

In [6]:
```
# Обучающая выборка:
train = pd.read_csv('Heart_Desease_Train.csv', sep=",")
# Тестовая выборка:
test = pd.read_csv('Heart_Desease_Test.csv', sep=",")
```

Проверим правильность создания обучающей и тестовой выборок:

In [7]:
```
train.head()
```

Out[7]:

	age	gender	chest_pain_type	blood_pressure	cholestoral	sugar	ECG	max_heart_rate	stenoca
0	61	1	0	148	203	0	1	161	
1	54	1	2	125	273	0	0	152	
2	71	0	2	110	265	1	0	130	
3	54	1	0	110	239	0	1	126	
4	66	1	0	112	212	0	0	132	

In [8]:
```
test.head()
```

Out[8]:

	age	gender	chest_pain_type	blood_pressure	cholestoral	sugar	ECG	max_heart_rate	stenoca
0	51	1	3	125	213	0	0	125	
1	51	0	2	130	256	0	0	149	
2	44	1	1	130	219	0	0	188	
3	56	1	0	130	283	1	0	103	
4	64	0	0	180	325	0	1	154	

2) Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.

Размер обучающего датасета - 212 строк на 14 столбцов, тестового - 91 строка на 14 столбцов:

In [9]:
```
train.shape, test.shape
```

Out[9]:
```
((212, 14), (91, 14))
```

Проверим, одинаковы ли типы данных в столбцах обучающего и тестового датасета:

In [10]:
```
train.dtypes
```

Out[10]:
```
age                 int64
gender              int64
chest_pain_type     int64
blood_pressure      int64
cholestoral         int64
sugar               int64
ECG                 int64
max_heart_rate      int64
stenocardia         int64
ST_depression     float64
slope               int64
vessels             int64
thal                int64
target              int64
dtype: object
```

In [11]:
```
test.dtypes
```

Out[11]:
```
age                 int64
gender              int64
chest_pain_type     int64
blood_pressure      int64
cholestoral         int64
sugar               int64
ECG                 int64
max_heart_rate      int64
stenocardia         int64
ST_depression     float64
slope               int64
vessels             int64
thal                int64
target              int64
dtype: object
```

Проверяем датасеты на наличие пустых значений:

In [12]:
```
train.isnull().sum()
```

Out[12]:
```
age                 0
gender              0
chest_pain_type     0
blood_pressure      0
cholestoral         0
sugar               0
ECG                 0
max_heart_rate      0
stenocardia         0
ST_depression       0
slope               0
vessels             0
thal                0
target              0
dtype: int64
```

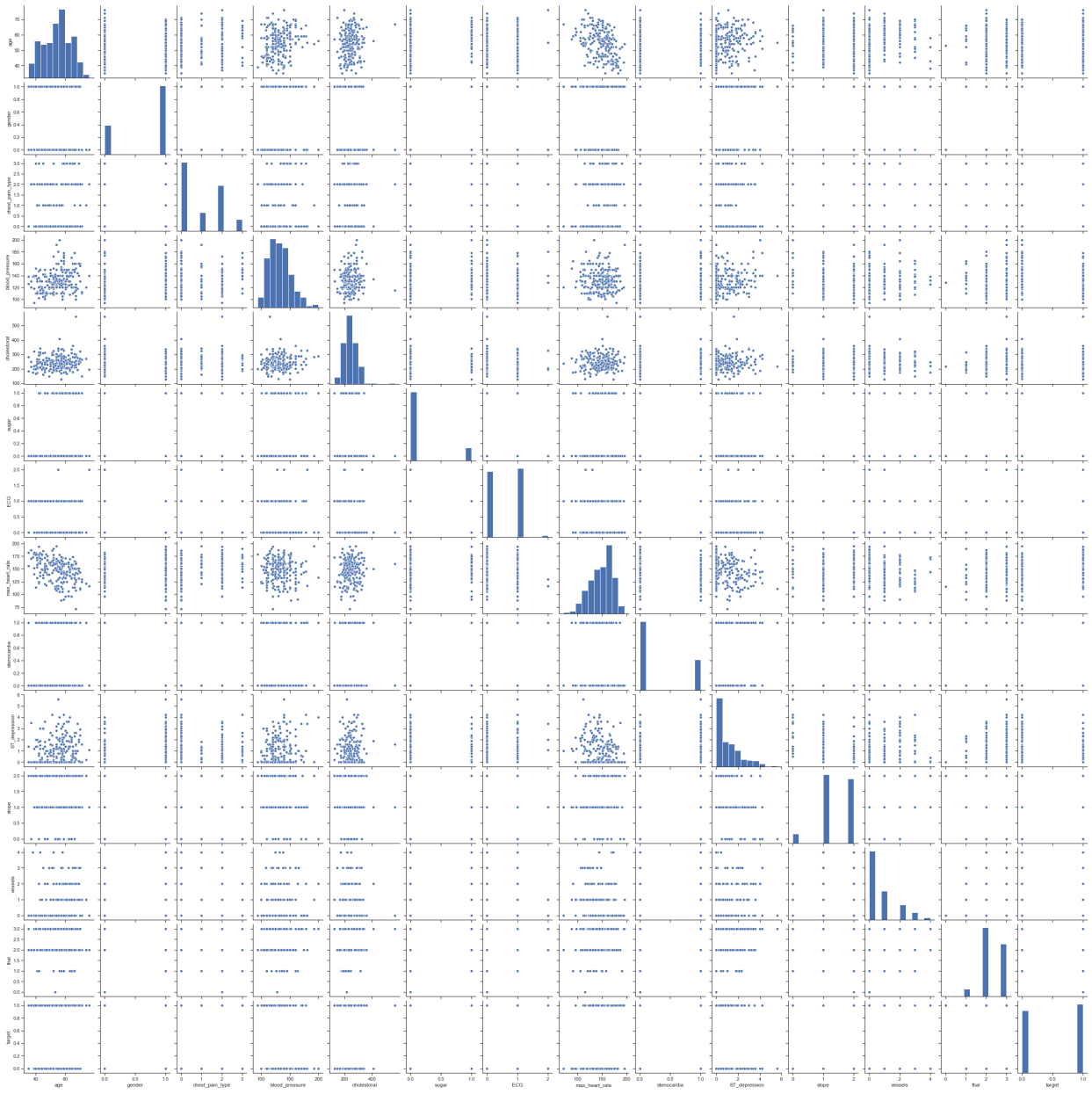
In [13]:
```
test.isnull().sum()
```

Out[13]:
```
age                 0
gender              0
chest_pain_type     0
blood_pressure      0
cholestoral         0
sugar               0
ECG                 0
max_heart_rate      0
stenocardia         0
ST_depression       0
slope               0
vessels             0
thal                0
target              0
dtype: int64
```

Построим парную диаграмму для наглядности структуры наших данных:

In [14]:
```
sns.pairplot(train)
```

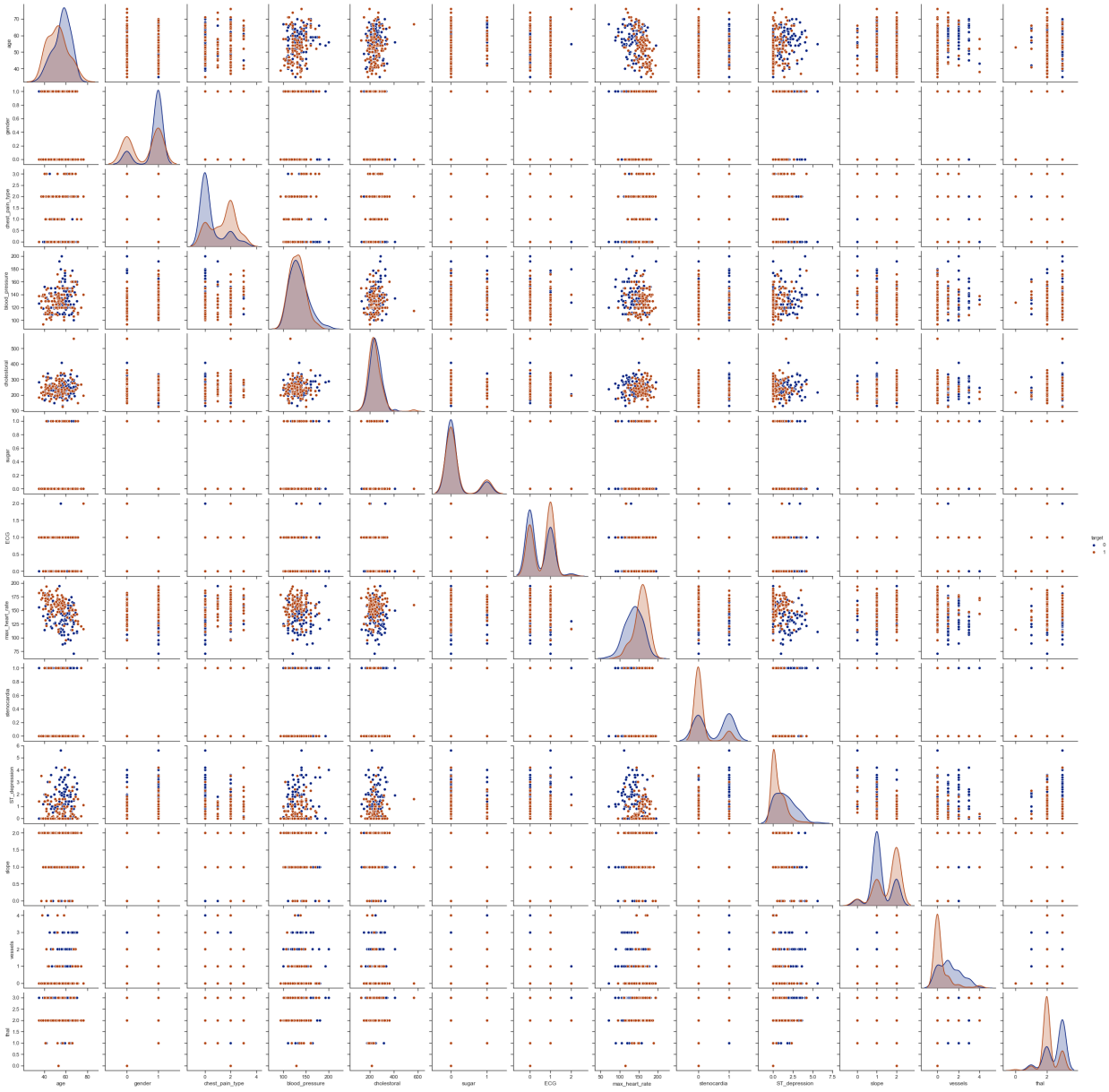
Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x1a1c9c2c90>
```



Посмотрим, насколько на эти зависимости влияет целевой признак:

In [15]:
```
sns.pairplot(train, hue="target", palette='dark')
```

Out[15]:
```
<seaborn.axisgrid.PairGrid at 0x1a22f57dd0>
```



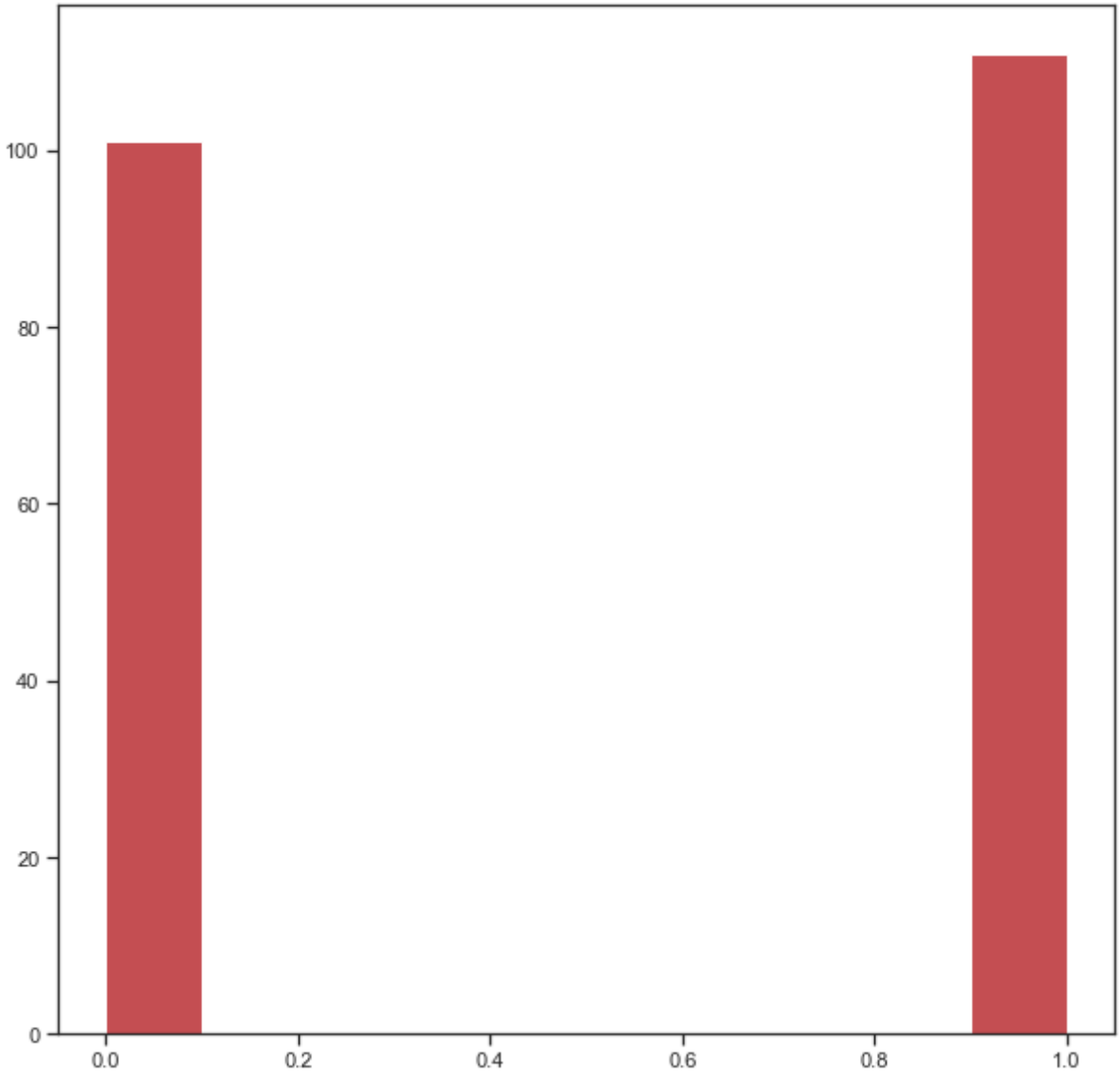
Убедимся, что целевой признак для задачи бинарной классификации в обучающем датасете содержит только 0 и 1:

In [16]:
```
train['target'].unique()
```

Out[16]:
```
array([0, 1])
```

Рассмотрим количество классов "0" и "1" целевого признака для задачи бинарной классификации в обучающем датасете:

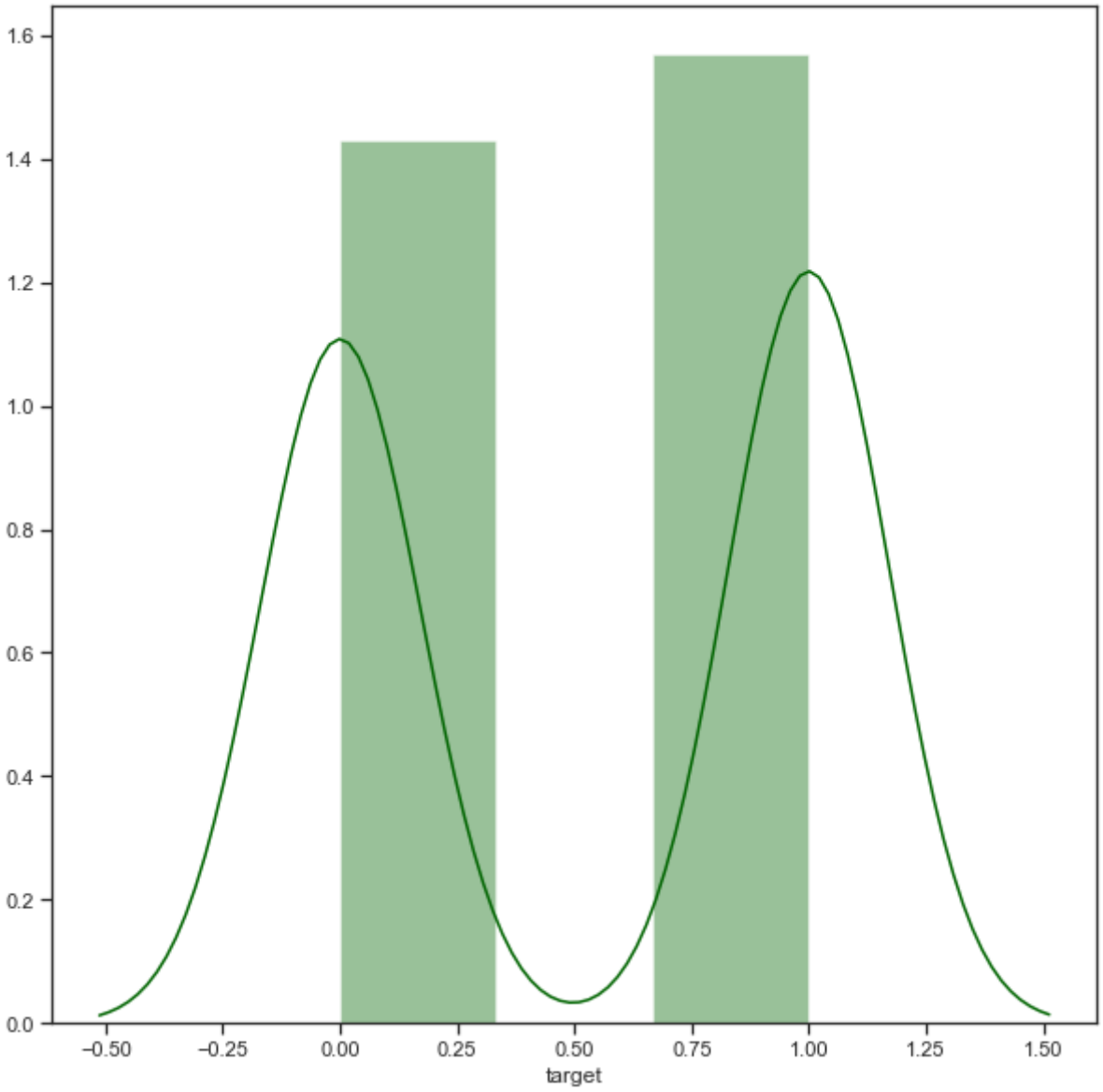
In [17]:
```
fig, ax = plt.subplots(figsize=(10,10))
plt.hist(train['target'], color="r")
plt.show()
```



Оценим здесь же плотность вероятности распределения:

In [18]:
```
fig, ax = plt.subplots(figsize=(10,10))
sns.distplot(train['target'], color="darkgreen")
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2b537310>



Подсчитаем дисбаланс классов для обучающей выборки:

In [19]:
```
total = train.shape[0]
class_0, class_1 = train['target'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100))
```

```
Класс 0 составляет 52.35999999999999%, а класс 1 составляет 47.64%.
```

In [20]:
```
train['target'].value_counts()
```

Out[20]:
```
1    111
0    101
Name: target, dtype: int64
```

Делаем вывод, что дисбаланс в обучающей выборке практически отсутствует.

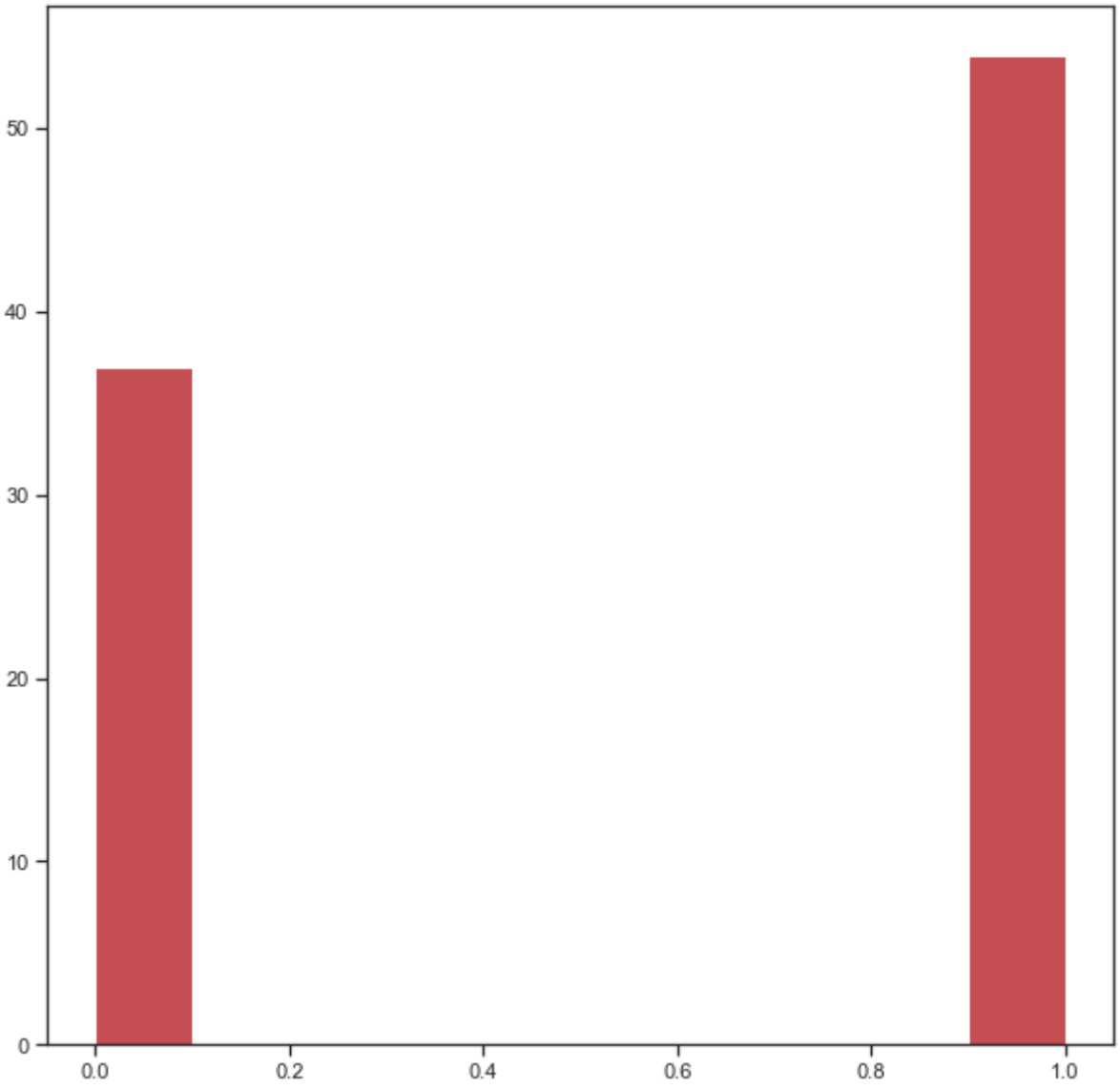
Убедимся, что целевой признак для задачи бинарной классификации в тестовом датасете содержит только 0 и 1:

In [21]:
```
test['target'].unique()
```

Out[21]:
```
array([1, 0])
```

Рассмотрим количество классов "0" и "1" целевого признака для задачи бинарной классификации в тестовом датасете:

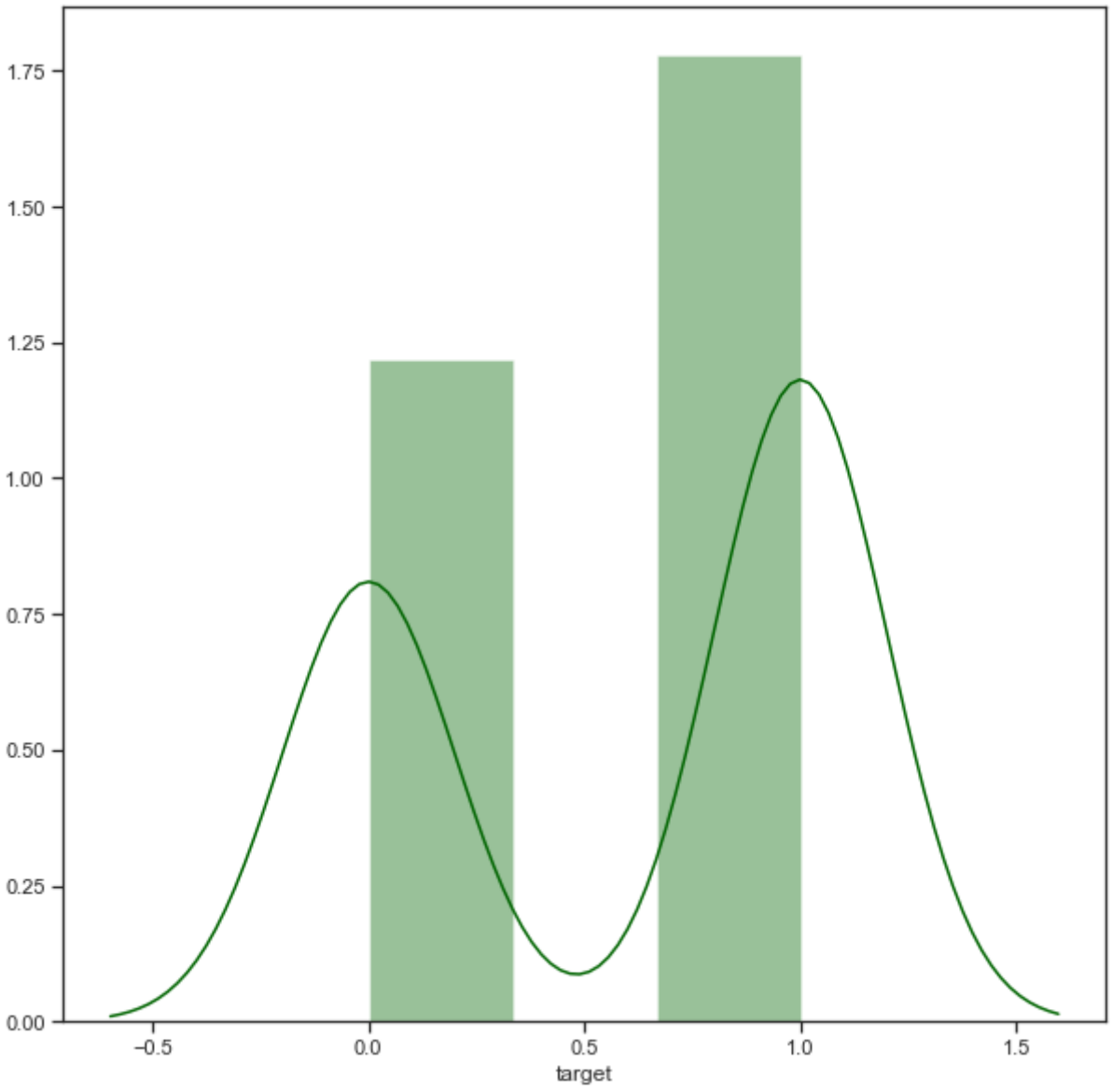
In [22]:
```
fig, ax = plt.subplots(figsize=(10,10))
plt.hist(test['target'], color="r")
plt.show()
```



Оценим здесь же плотность вероятности распределения:

In [23]:
```
fig, ax = plt.subplots(figsize=(10,10))
sns.distplot(test['target'], color="darkgreen")
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2cb8e3d0>
```



Подсчитаем дисбаланс классов для обучающей выборки:

In [24]:
```
total = test.shape[0]
class_0, class_1 = test['target'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100))
```

```
Класс 0 составляет 59.34%, а класс 1 составляет 40.660000000000004%.
```

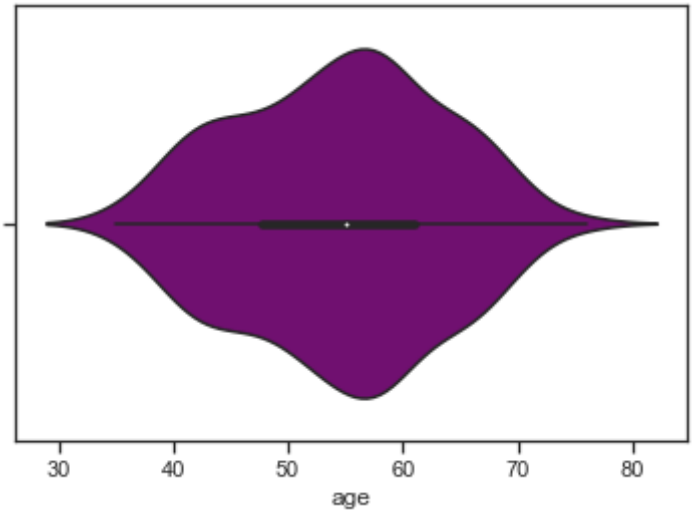
In [25]:
```
test['target'].value_counts()
```

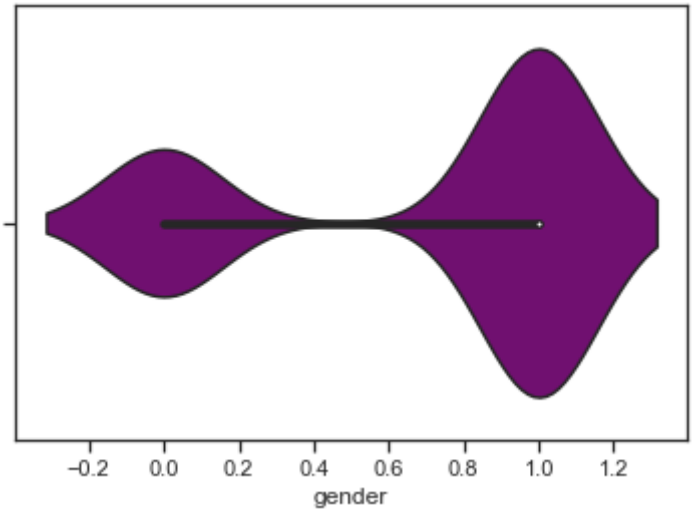
Out[25]:
```
1    54
0    37
Name: target, dtype: int64
```

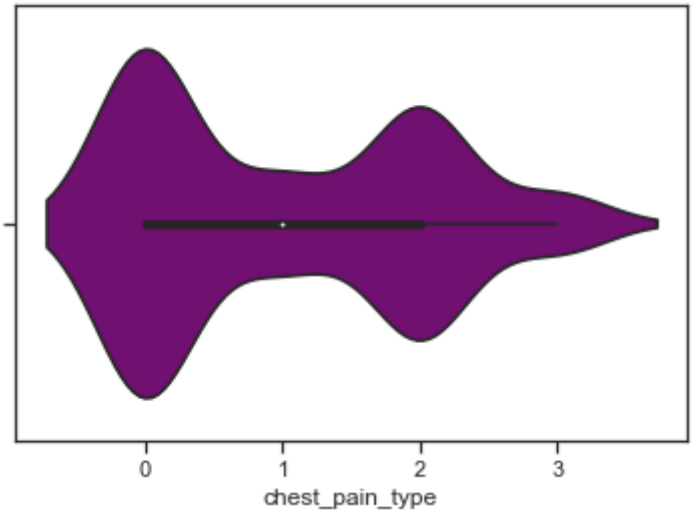
Делаем вывод, что в тестовой выборке дисбаланс присутствует, но является приемлемым

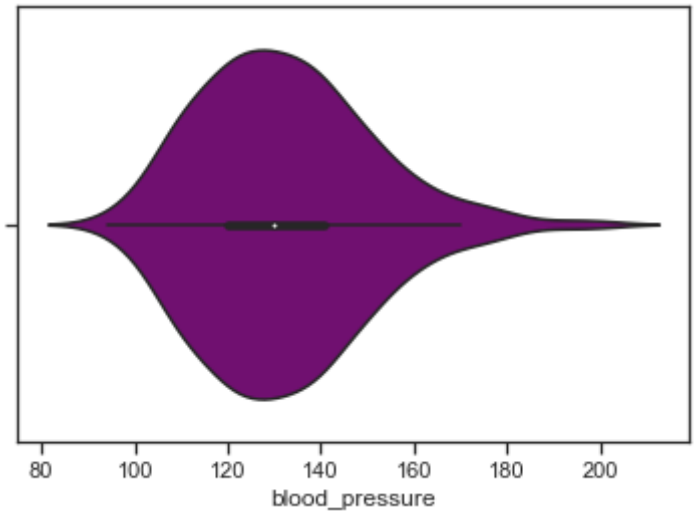
Рассмотрим скрипичные диаграммы для числовых столбцов обучающей выборки (отображают одномерное распределение вероятности и распределение плотности):

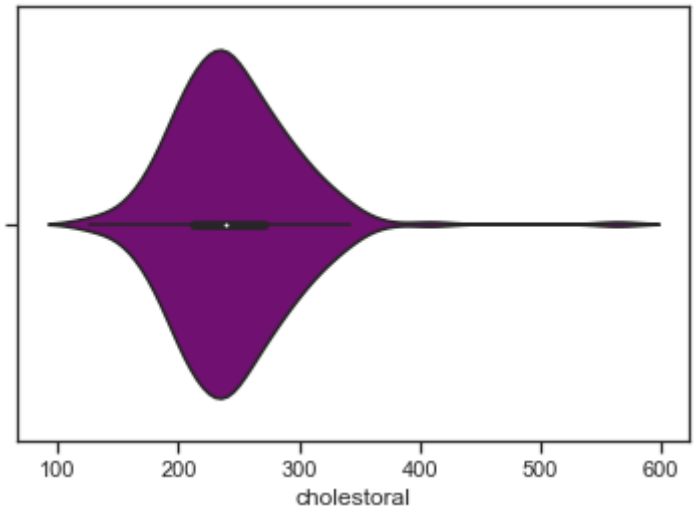
In [26]:
```
for col in ['age', 'gender', 'chest_pain_type', 'blood_pressure', 'cholesto
       'sugar', 'ECG', 'max_heart_rate', 'stenocardia', 'ST_depression',
       'slope', 'vessels', 'thal', 'target']:
    sns.violinplot(x=train[col], color="purple")
    plt.show()
```

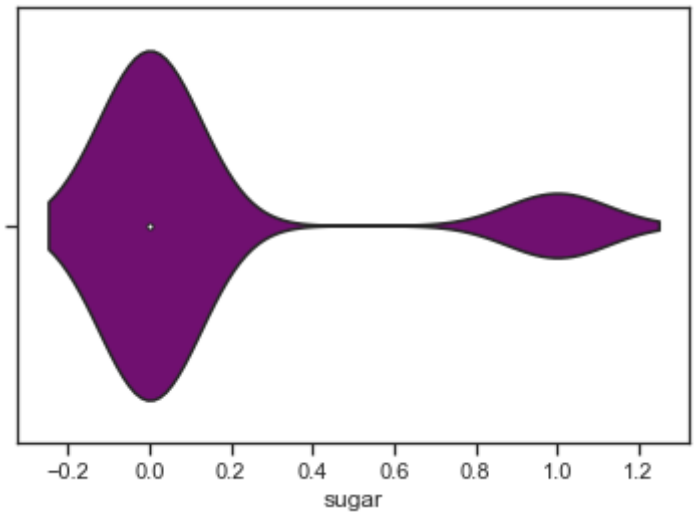


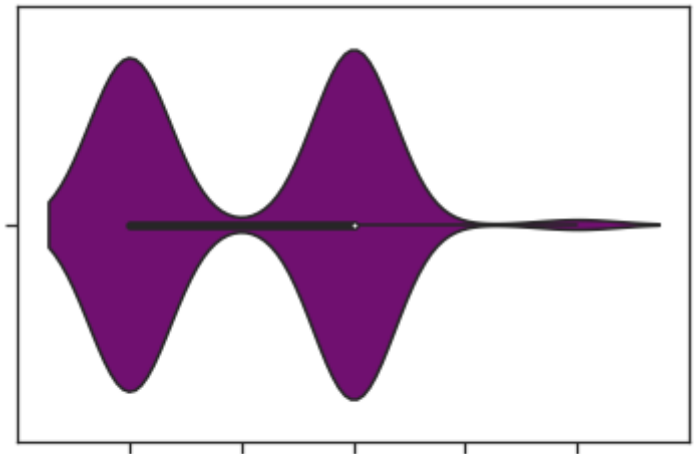


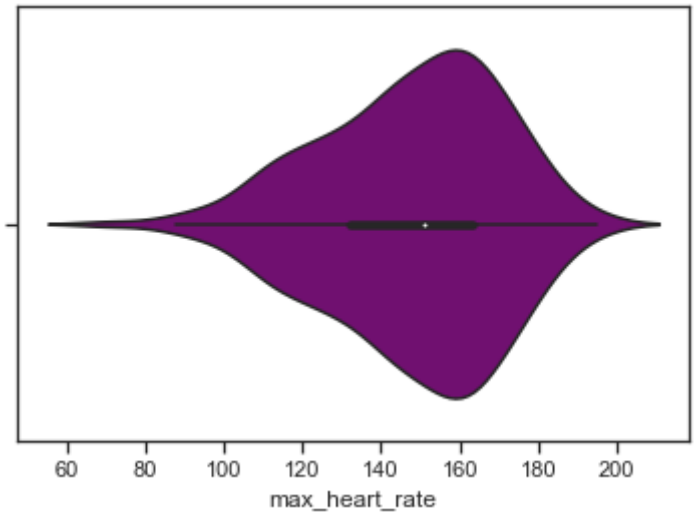


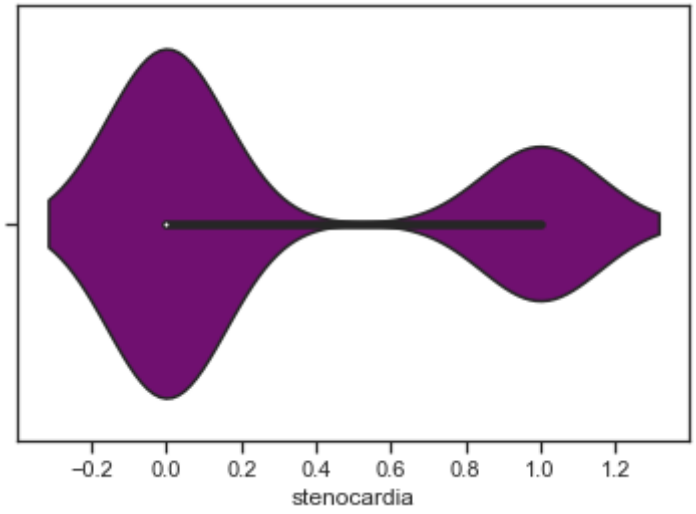

80
100
120
140
160
180
200
blood_pressure

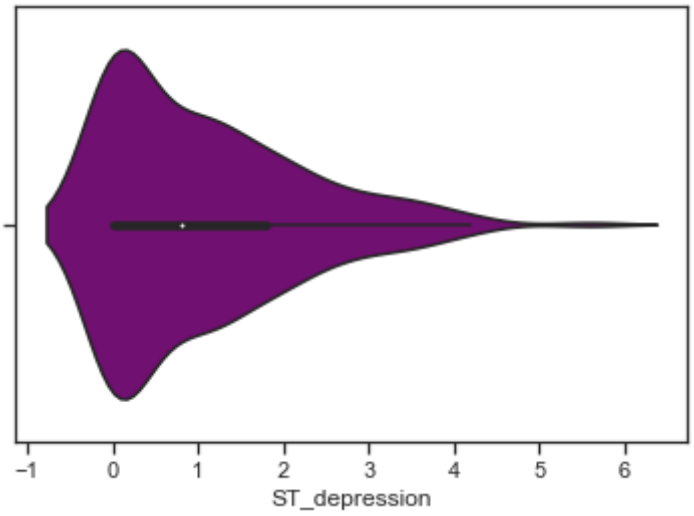

100
200
300
400
500
600
cholestoral

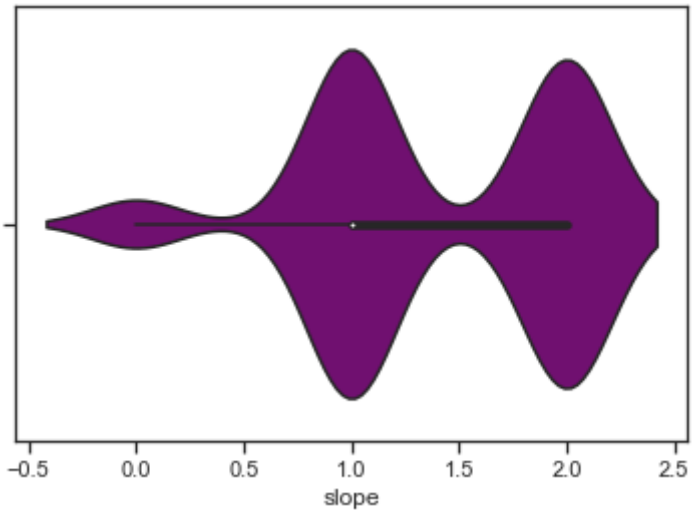

−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
sugar

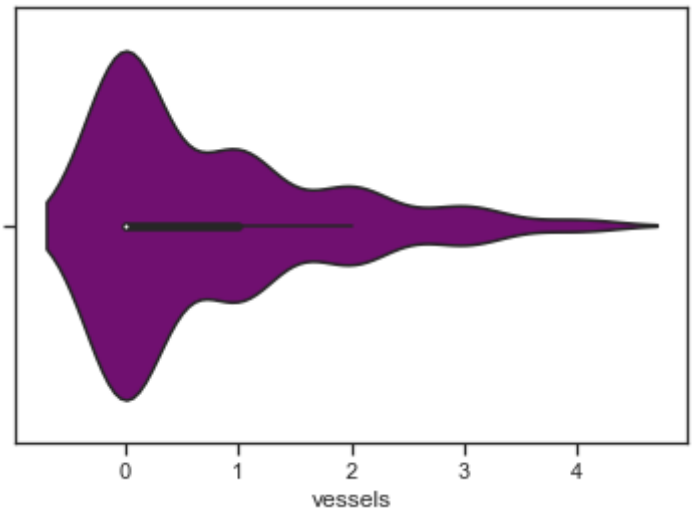


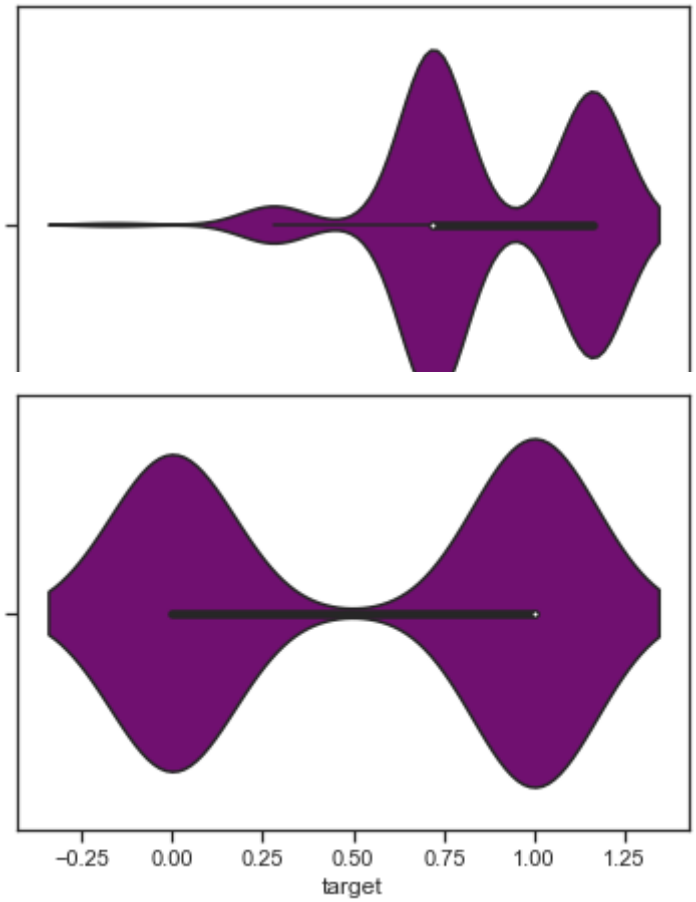

60
80
100
120
140
160
180
200
max_heart_rate


−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
stenocardia


−1
0
1
2
3
4
5
6
ST_depression


−0.5
0.0
0.5
1.0
1.5
2.0
2.5
slope


0
1
2
3
4
vessels



3) Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

In [27]:
```
train.dtypes
```

Out[27]:
```
age                 int64
gender              int64
chest_pain_type     int64
blood_pressure      int64
cholestoral         int64
sugar               int64
ECG                 int64
max_heart_rate      int64
stenocardia         int64
ST_depression     float64
slope               int64
vessels             int64
thal                int64
target              int64
dtype: object
```

Кодирование признаков не требуется, поскольку все данные представлены в числовом виде. Для построения моделей будем использовать все признаки. Объединим обучающую и тестовую выборки для масштабирования данных. Для начала создадим вспомогательные колонки для возможности дальнейшего разделения целого датасета:

In [28]:
```
train['dataset'] = 'TRAIN'
test['dataset'] = 'TEST'
```

Выберем столбцы для объединения датасетов:

In [29]:
```
join_cols = ['age', 'gender', 'chest_pain_type', 'blood_pressure', 'cholest
       'sugar', 'ECG', 'max_heart_rate', 'stenocardia', 'ST_depression',
       'slope', 'vessels', 'thal', 'target', 'dataset']
```

In [30]:
```
data_all = pd.concat([train[join_cols], test[join_cols]])
```

Проверяем корректность объединения:

In [31]:
```
assert data_all.shape[0] == train.shape[0]+test.shape[0]
```

In [32]:
```
data_all.head()
```

Out[32]:

	age	gender	chest_pain_type	blood_pressure	cholestoral	sugar	ECG	max_heart_rate	stenoca
0	61	1	0	148	203	0	1	161	
1	54	1	2	125	273	0	0	152	
2	71	0	2	110	265	1	0	130	
3	54	1	0	110	239	0	1	126	
4	66	1	0	112	212	0	0	132	

Выберем столбцы для масштабирования:

In [33]:
```
 'cholestoral', 'sugar', 'ECG', 'max_heart_rate', 'stenocardia', 'ST_depres
```

In [34]:
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data_all[scale_cols])
```

Добавляем масштабированные данные в наш датасет:

In [35]:
```
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data_all[new_col_name] = sc1_data[:,i]
```

Проверяем корректность:

In [36]:
```
data_all.head()
```

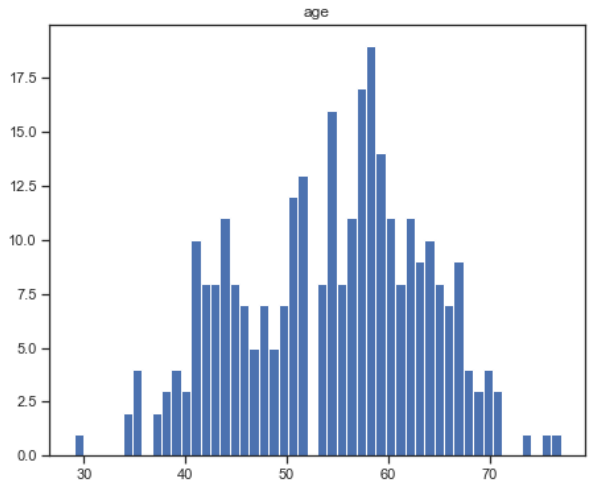
Out[36]:

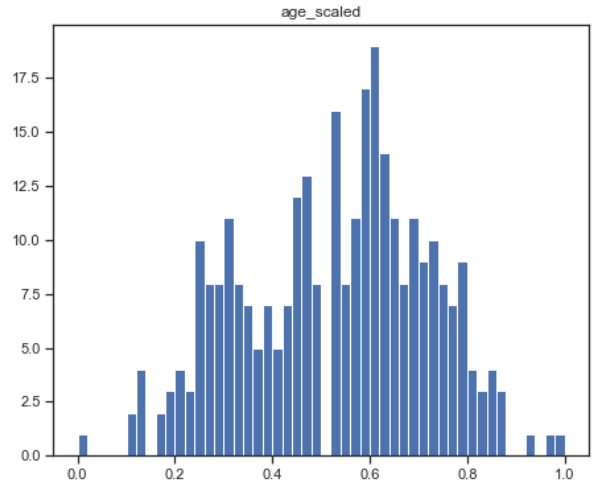
	age	gender	chest_pain_type	blood_pressure	cholestoral	sugar	ECG	max_heart_rate	stenoca
0	61	1	0	148	203	0	1	161	
1	54	1	2	125	273	0	0	152	
2	71	0	2	110	265	1	0	130	
3	54	1	0	110	239	0	1	126	
4	66	1	0	112	212	0	0	132	

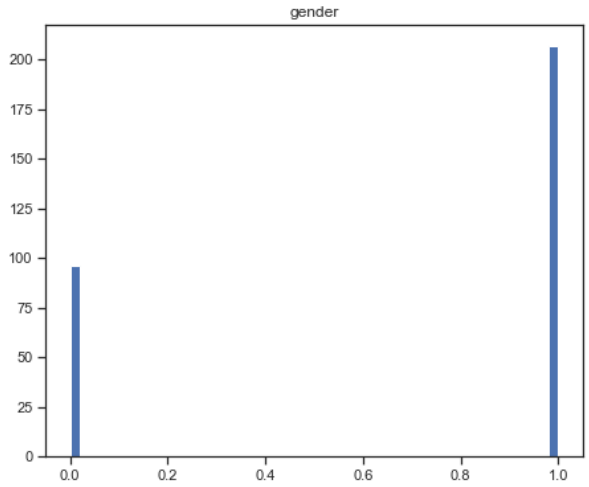
5 rows × 29 columns

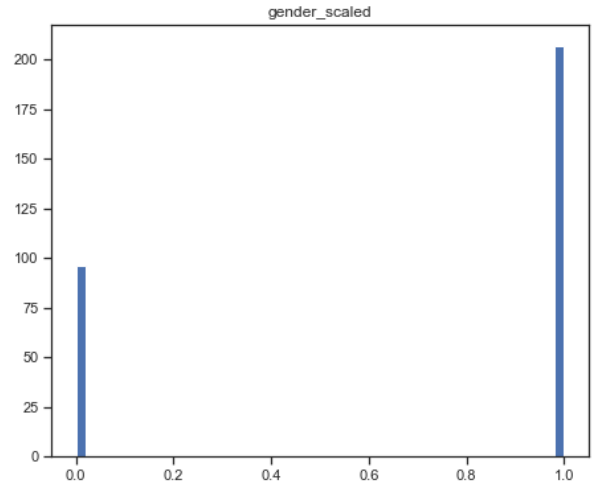
Посмотрим, повлияло ли масштабирование на распределение данных:

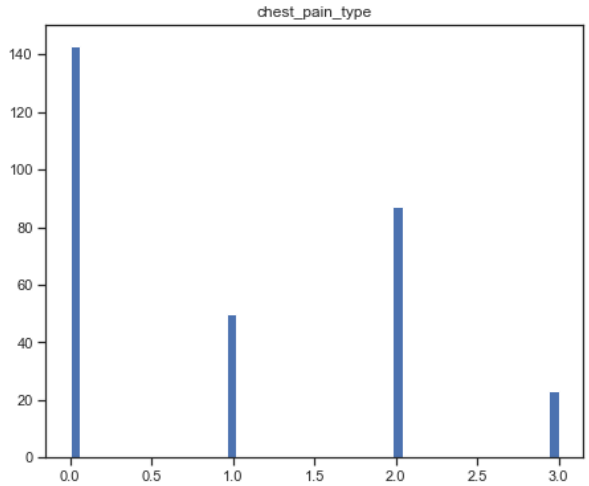
In [37]:
```
for col in scale_cols:
    col_scaled = col + '_scaled'
    fig, ax = plt.subplots(1, 2, figsize=(16,6))
    ax[0].hist(data_all[col], 50)
    ax[1].hist(data_all[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```

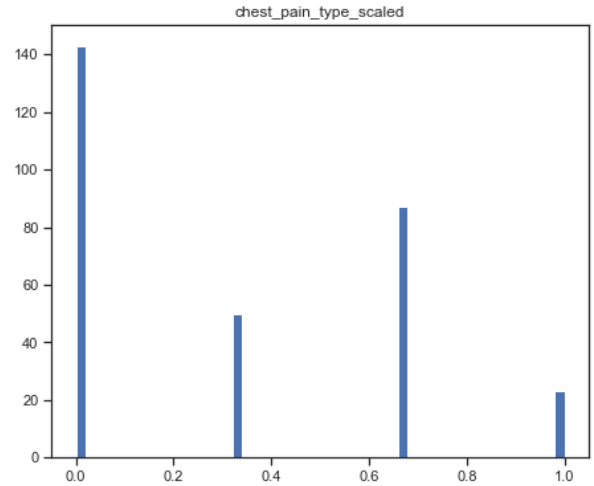


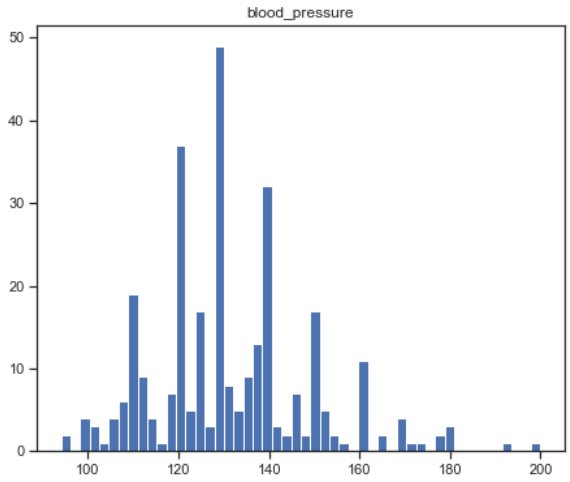


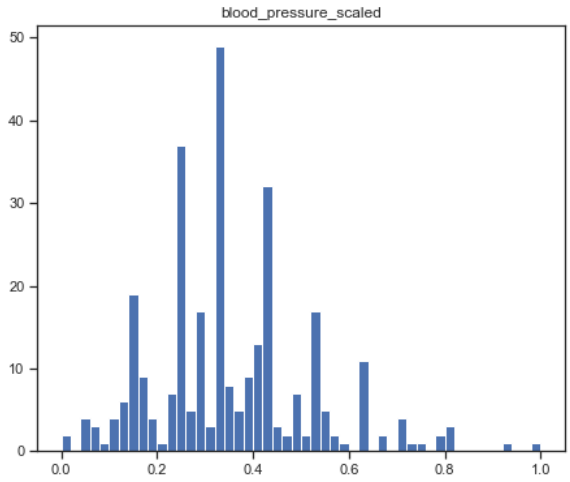


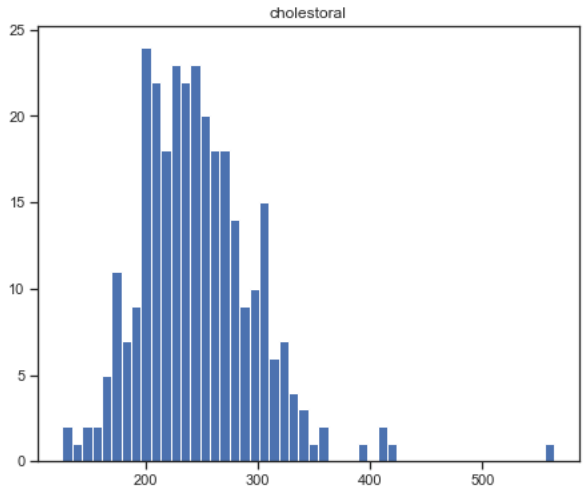


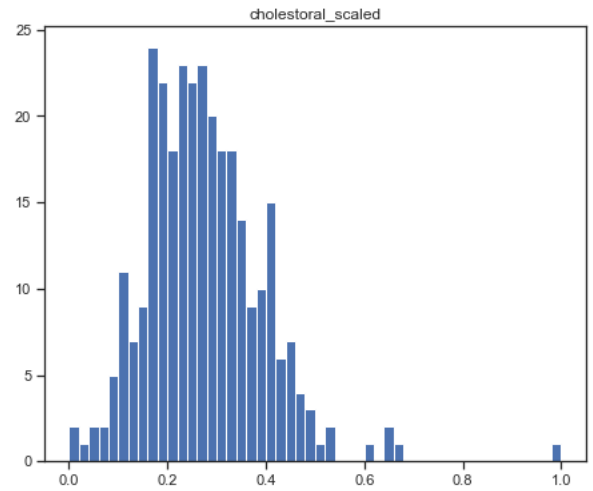


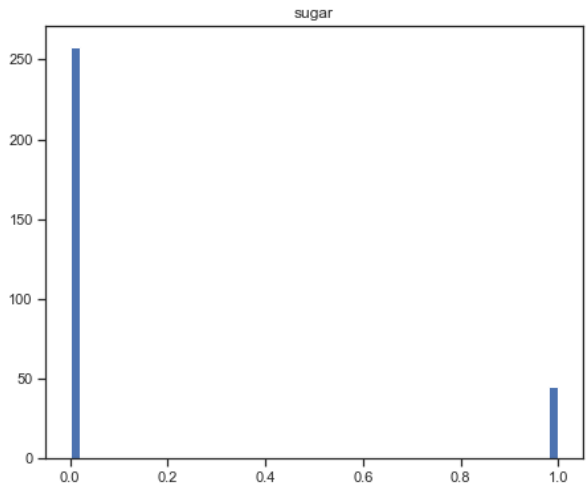


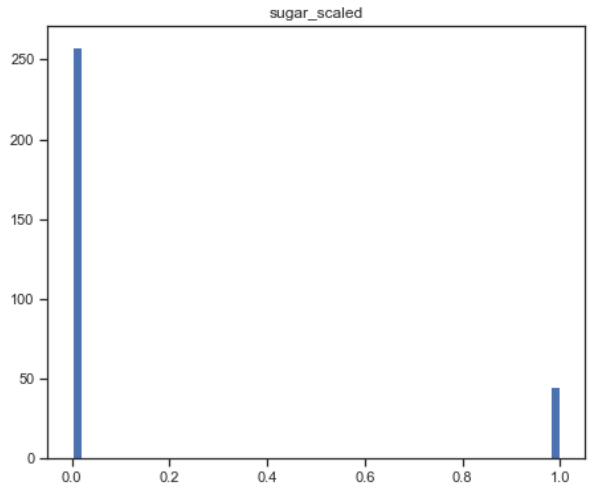

blood_pressure
50
40
30
20
10
0
100
120
140
160
180
200


blood_pressure_scaled
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0

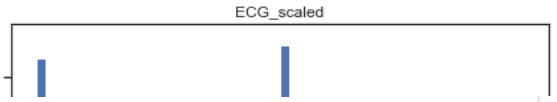

cholestoral
25
20
15
10
5
0
200
300
400
500

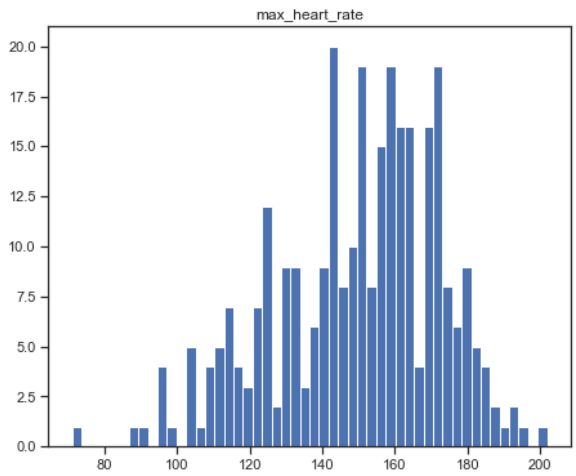

cholestoral_scaled
25
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0

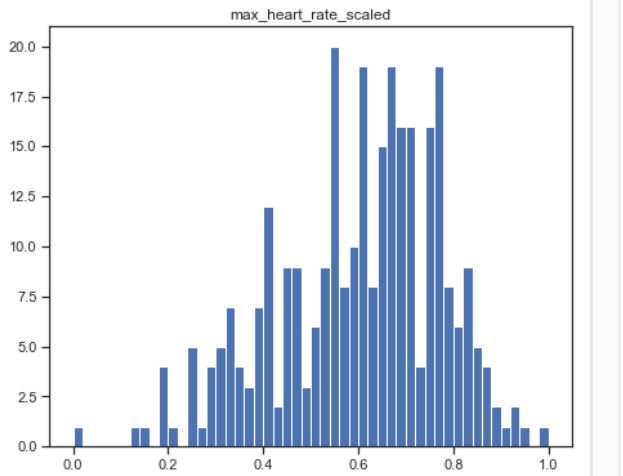

sugar
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0

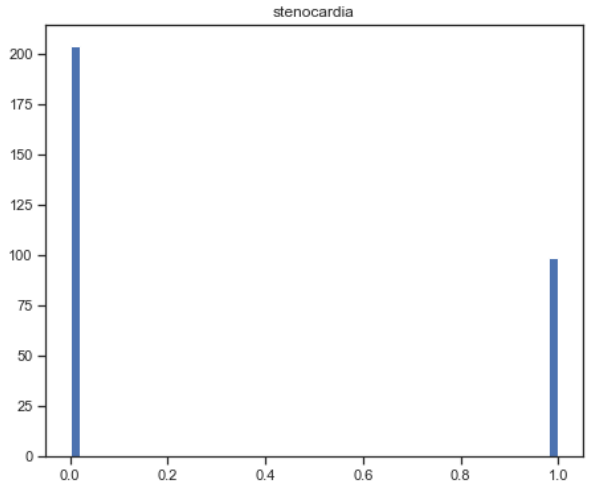

sugar_scaled
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0

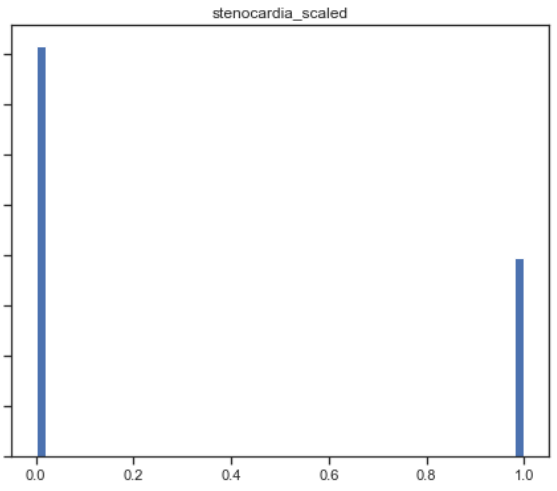

ECG
140

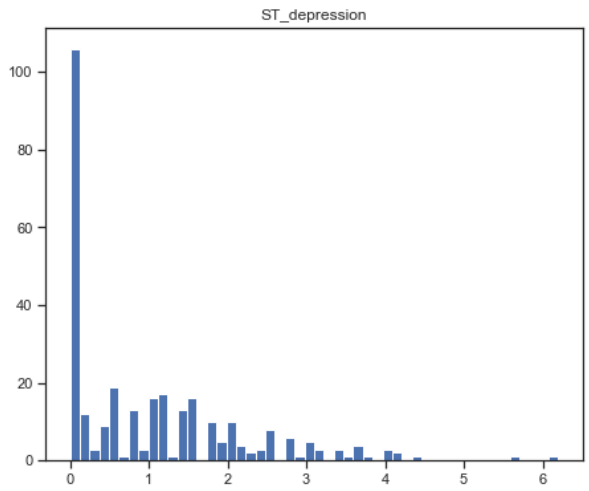

ECG_scaled
140

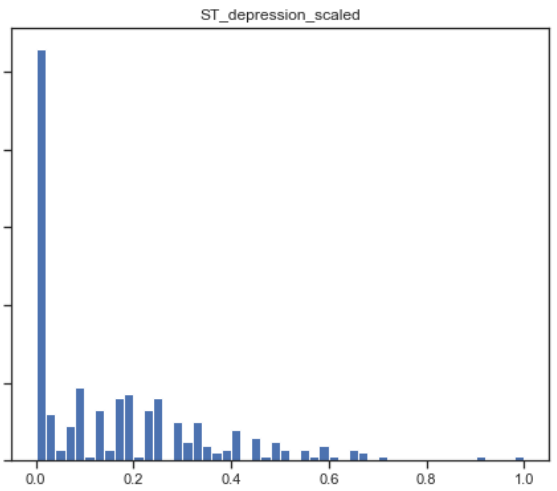

max_heart_rate
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
80
100
120
140
160
180
200

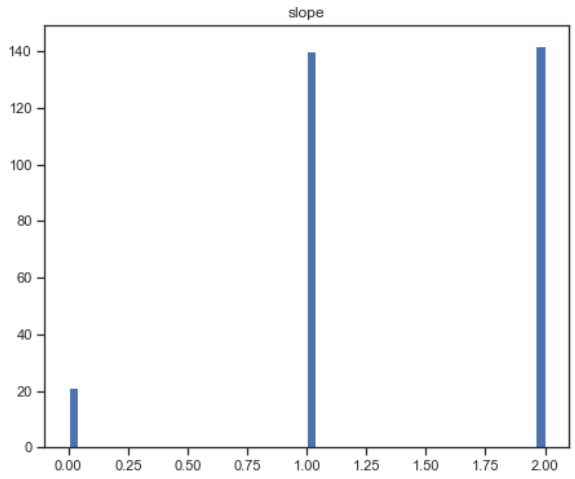

max_heart_rate_scaled
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

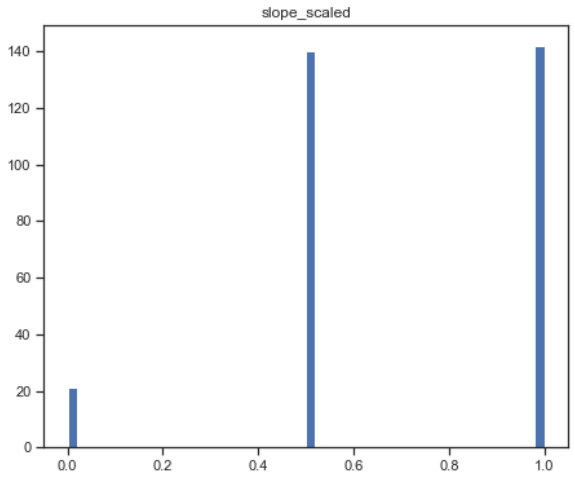

stenocardia
200
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0

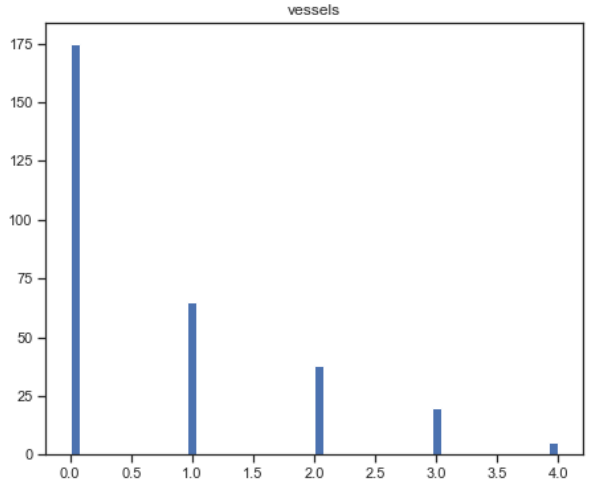

stenocardia_scaled
200
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0

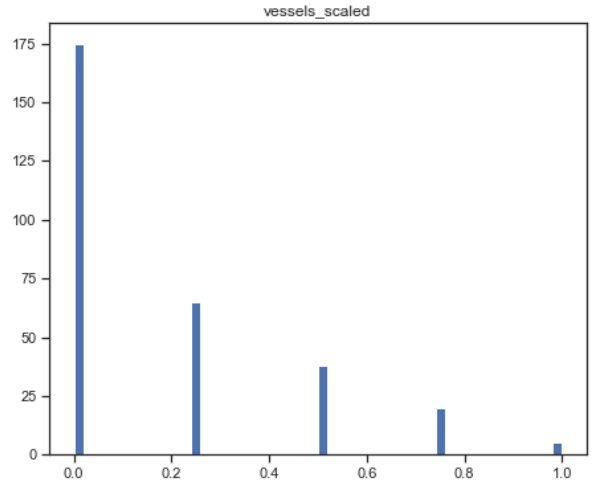

ST_depression
100
80
60
40
20
0
0
1
2
3
4
5
6

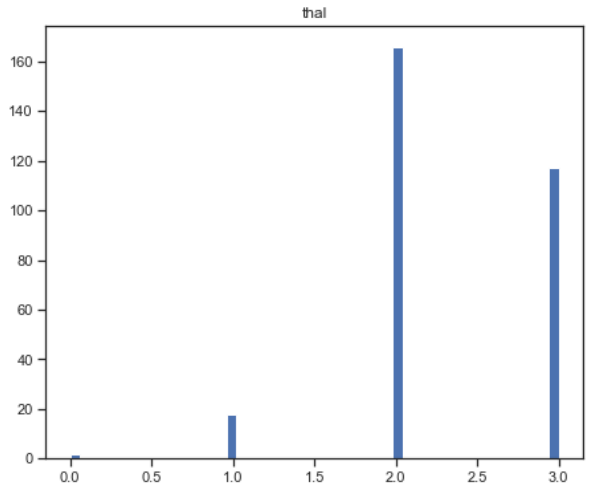

ST_depression_scaled
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

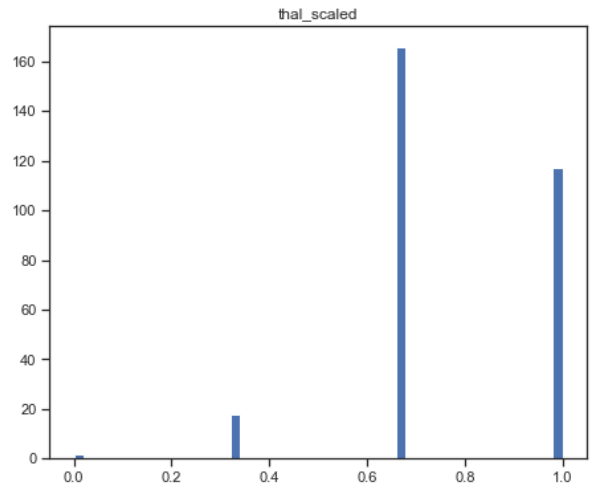

slope
140
120
100
80
60
40
20
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

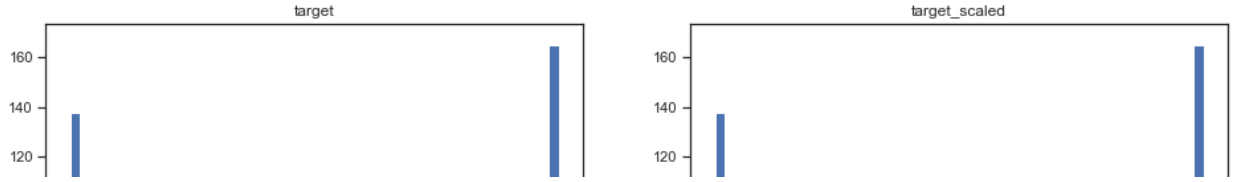

slope_scaled
140
120
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0


vessels
175
150
125
100
75
50
25
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


vessels_scaled
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0


thal
160
140
120
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


thal_scaled
160
140
120
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0



Выводы: масштабирование данных не повлияло на их распределение

4) Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

Включим тестовую выборку в корреляционную матрицу:

In [38]:
```
corr_cols_1 = scale_cols + ['target']
corr_cols_1
```

Out[38]:
```
['age',
 'gender',
 'chest_pain_type',
 'blood_pressure',
 'cholestoral',
 'sugar',
 'ECG',
 'max_heart_rate',
 'stenocardia',
 'ST_depression',
 'slope',
 'vessels',
 'thal',
 'target',
 'target']
```

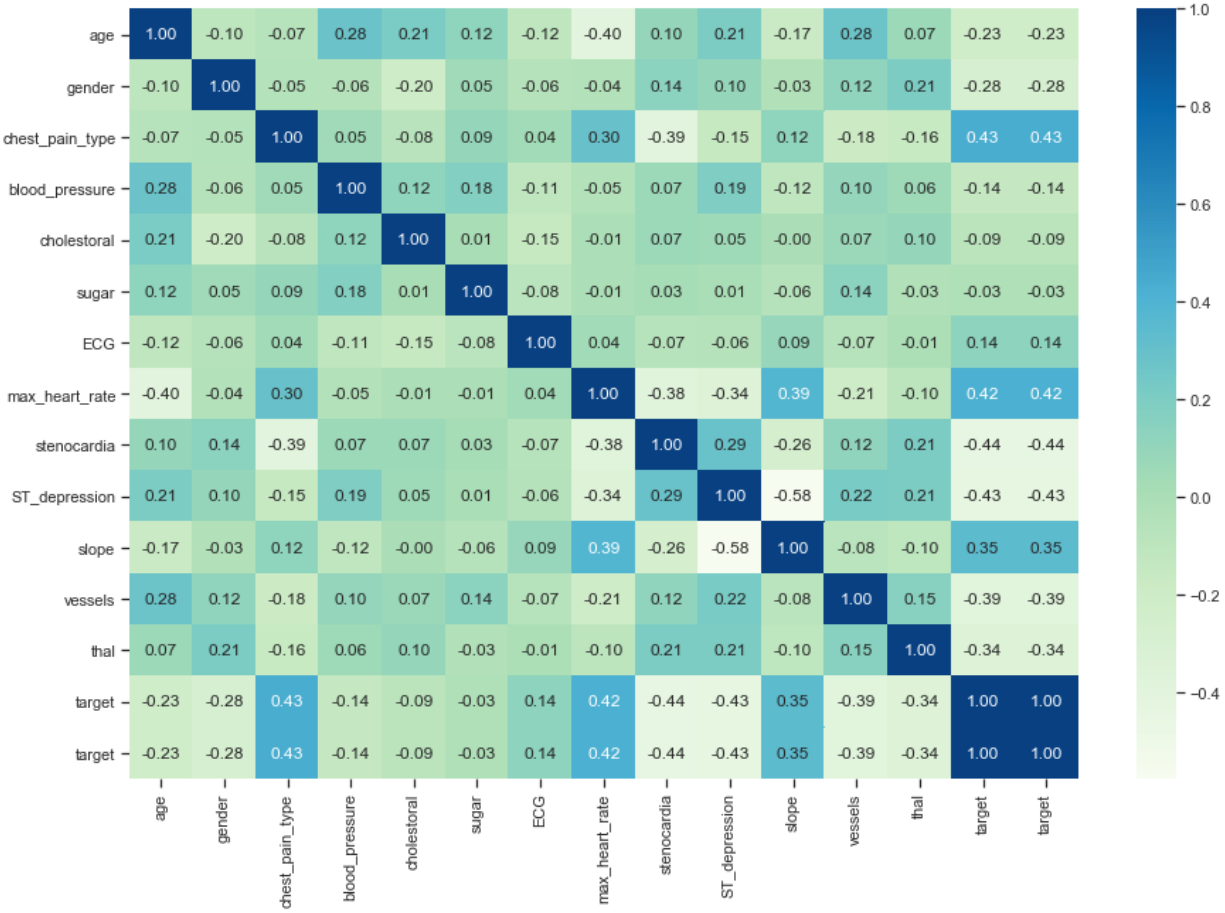
In [39]:
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['target']
corr_cols_2
```

Out[39]:
```
['age_scaled',
 'gender_scaled',
 'chest_pain_type_scaled',
 'blood_pressure_scaled',
 'cholestoral_scaled',
 'sugar_scaled',
 'ECG_scaled',
 'max_heart_rate_scaled',
 'stenocardia_scaled',
 'ST_depression_scaled',
 'slope_scaled',
 'vessels_scaled',
 'thal_scaled',
 'target_scaled',
 'target']
```

Построим корреляционную матрицу:

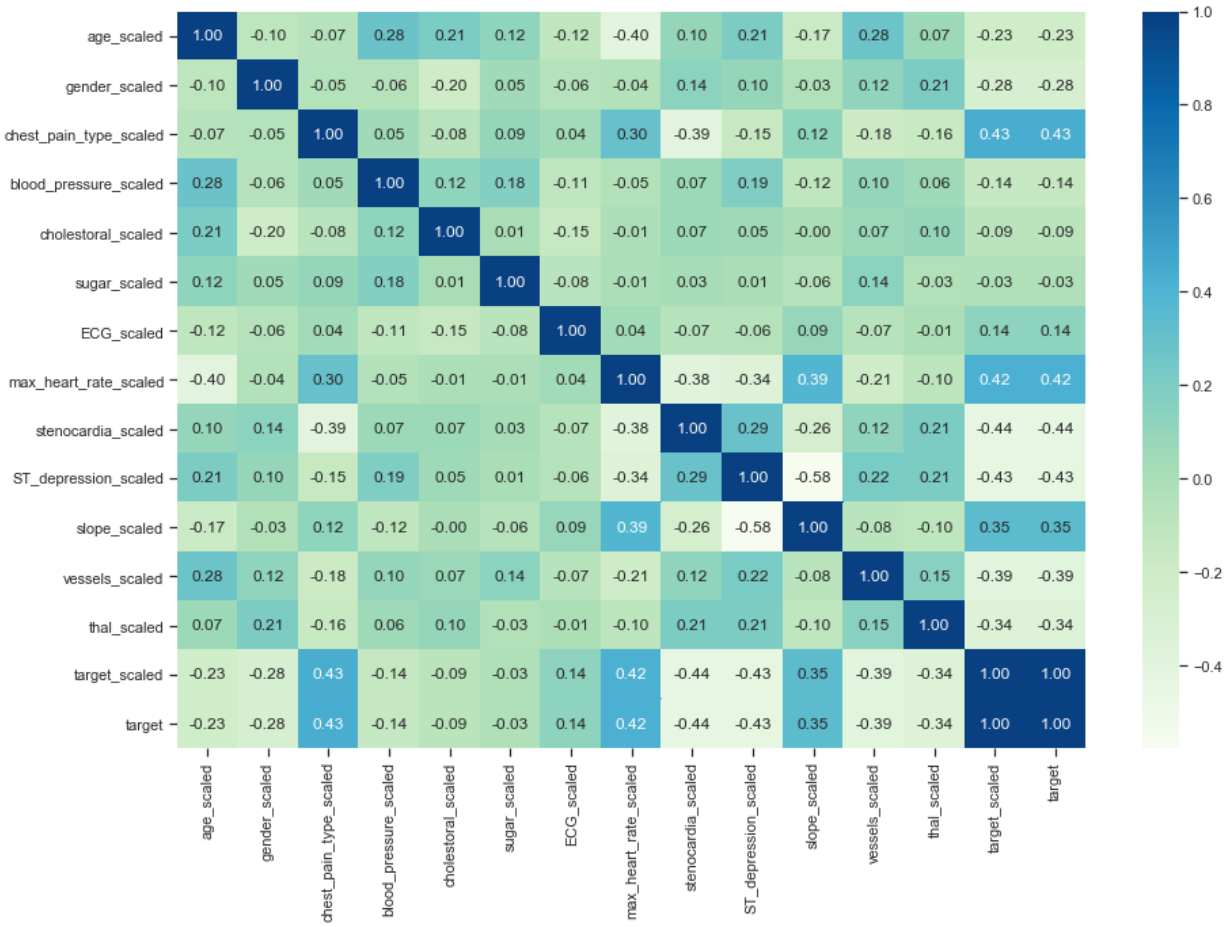
In [40]:
```
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(data_all[corr_cols_1].corr(), annot=True, fmt='.2f', cmap='GnBu
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2f459090>
```



In [41]:
```
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(data_all[corr_cols_2].corr(), annot=True, fmt='.2f', cmap="GnBu
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2eb94350>



Корреляционные матрицы для исходных и масштабированных данных полностью

совпадают.

Целевой признак решения задачи классификации target наиболее коррелирует с (по убыванию): stenocardia, ST_depression, chest_pain_type, max_heart_rate, vessels, slope, thal и gender. Признак max_heart_rate коррелирует с большей частью остальных выше перечисленных признаков. Поэтому в модель классификации войдут все выше перечисленные признаки за исключением max_heart_rate.

Целевой признак решения задачи регрессии max_heart_rate наиболее коррелирует с (по убыванию): target, age, slope, stenocardia, ST_depression, chest_pain_type и vessels. Однако большая часть из этих признаков коррелирует друг с другом. Опытным путем выявлено, что оптимальным набором для модели регрессии будут признаки target, slope и vessels.

5) Выбор метрик для последующей оценки качества моделей.

В качестве метрик для решения задачи классификации будем использовать:

- Метрика precision: $precision = \frac{TP}{TP+FP}$
- Метрика recall (полнота): $recall = \frac{TP}{TP+FN}$
- Метрика F_1-мера: $F_\beta = (1+\beta^2) \cdot \frac{precision \cdot recall}{precision+recall}$, где β определяет вес точности в метрике.
- Метрика ROC AUC: $TPR = \frac{TP}{TP+FN}$ - True Positive Rate, откладывается по оси ординат. Совпадает с recall. $FPR = \frac{FP}{FP+TN}$ - False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно.

В качестве метрик для решения задачи регрессии будем использовать:

- Mean absolute error - средняя абсолютная ошибка $MAE(y, \hat{y}) = \frac{1}{N} \cdot \sum_{i=1}^{N} |y_i - \hat{y_i}|$ где: y - истинное значение целевого признака, $\hat{y}$ - предсказанное значение целевого признака, N - размер тестовой выборки
- Mean squared error - средняя квадратичная ошибка $MSE(y, \hat{y}) = \frac{1}{N} \cdot \sum_{i=1}^{N} (y_i - \hat{y_i})^2$, где: y - истинное значение целевого признака, $\hat{y}$ - предсказанное значение целевого признака, N - размер тестовой выборки
- Метрика R^2 или коэффициент детерминации $R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y_i})^2}{\sum_{i=1}^{N} (y_i - \overline{y_i})^2}$, где: y - истинное значение целевого признака, $\hat{y}$ - предсказанное значение целевого признака, N - размер тестовой выборки, $\overline{y_i} = \frac{1}{N} \cdot \sum_{i=1}^{N} y_i$

Введем класс, который позволит сохранять метрики качества построенных моделей и реализует визуализацию метрик качества:

In [42]:
```
class MetricLogger:

    def __init__(self):
        self.df = pd.DataFrame(
            {'metric': pd.Series([], dtype='str'),
            'alg': pd.Series([], dtype='str'),
            'value': pd.Series([], dtype='float')})

    def add(self, metric, alg, value):
        """
        Добавление значения
        """
        # Удаление значения если оно уже было ранее добавлено
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==a
        # Добавление нового значения
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
        self.df = self.df.append(temp, ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        """
        Формирование данных с фильтром по метрике
        """
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        """
        Вывод графика
        """
        array_labels, array_metric = self.get_data_for_metric(metric, ascen
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barh(pos, array_metric,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()
```

6) Выбор наиболее подходящих моделей для решения задачи классификации и регрессии.

Для задачи классификации будем использовать следующие модели:

- Логистическая регрессия
- Метод ближайших соседей

- Машина опорных векторов
- Решающее дерево
- Случайный лес
- Градиентный бустинг

Для задачи регрессии будем использовать следующие модели:

- Линейная регрессия
- Метод ближайших соседей
- Машина опорных векторов
- Решающее дерево
- Случайный лес
- Градиентный бустинг

7) Формирование обучающей и тестовой выборок на основе исходного набора данных.

Выделим обучающую и тестовую выборки на основе масштабированных данных с помощью фильтра:

In [43]:
```
train_data_all = data_all[data_all['dataset']=='TRAIN']
test_data_all = data_all[data_all['dataset']=='TEST']
train_data_all.shape, test_data_all.shape
```

Out[43]:
```
((212, 29), (91, 29))
```

Определим признаки для задачи классификации:

In [44]:
```
task_clas_cols = ['stenocardia', 'gender', 'slope', 'ST_depression', 'chest
```

Определим выборки для задачи классификации:

In [45]:
```
clas_X_train = train_data_all[task_clas_cols]
clas_X_test = test_data_all[task_clas_cols]
clas_Y_train = train_data_all['target']
clas_Y_test = test_data_all['target']
clas_X_train.shape, clas_X_test.shape, clas_Y_train.shape, clas_Y_test.shap
```

Out[45]:
```
((212, 7), (91, 7), (212,), (91,))
```

Определим признаки для задачи регрессии:

In [134]:
```
task_regr_cols = ['slope', 'vessels', 'chest_pain_type', 'target']
```

Определим выборки для задачи регресии:

In [135]:
```
regr_X_train = train_data_all[task_regr_cols]
regr_X_test = test_data_all[task_regr_cols]
regr_Y_train = train_data_all['max_heart_rate']
regr_Y_test = test_data_all['max_heart_rate']
regr_X_train.shape, regr_X_test.shape, regr_Y_train.shape, regr_Y_test.shap
```

Out[135]:
```
((212, 4), (91, 4), (212,), (91,))
```

8) Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.

8.1) Решение задачи классификации

Определим модели:

In [136]:
```
clas_models = {'LogR': LogisticRegression(),
               'KNN_5':KNeighborsClassifier(n_neighbors=5),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

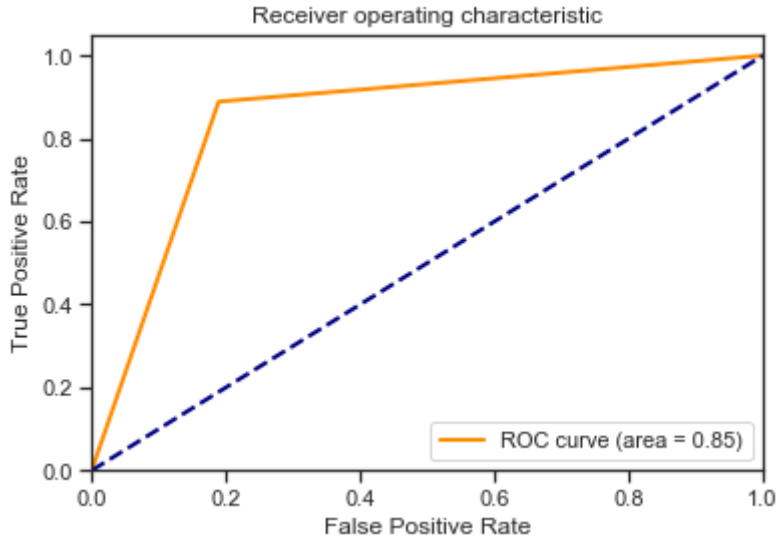
Сохранение метрик:

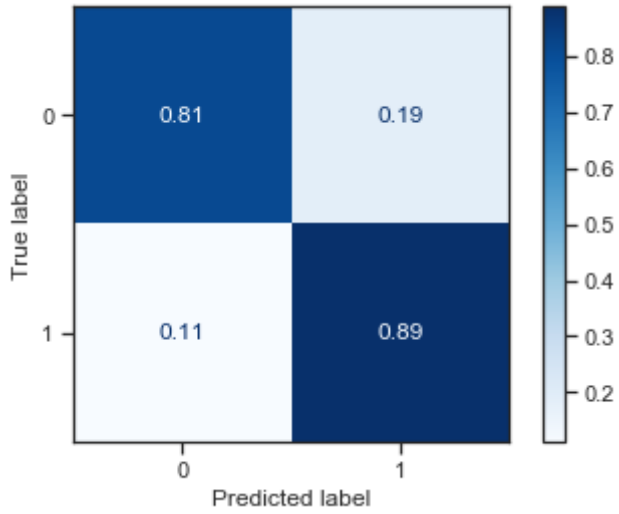
In [137]:
```
clasMetricLogger = MetricLogger()
```

In [138]:
```
def clas_train_model(model_name, model, clasMetricLogger):
    model.fit(clas_X_train, clas_Y_train)
    Y_pred = model.predict(clas_X_test)
    precision = precision_score(clas_Y_test.values, Y_pred)
    recall = recall_score(clas_Y_test.values, Y_pred)
    f1 = f1_score(clas_Y_test.values, Y_pred)
    roc_auc = roc_auc_score(clas_Y_test.values, Y_pred)

    clasMetricLogger.add('precision', model_name, precision)
    clasMetricLogger.add('recall', model_name, recall)
    clasMetricLogger.add('f1', model_name, f1)
    clasMetricLogger.add('roc_auc', model_name, roc_auc)

    print('****************************************************')
    print(model)
    print('****************************************************')
    draw_roc_curve(clas_Y_test.values, Y_pred)

    plot_confusion_matrix(model, clas_X_test, clas_Y_test.values,
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    plt.show()
```

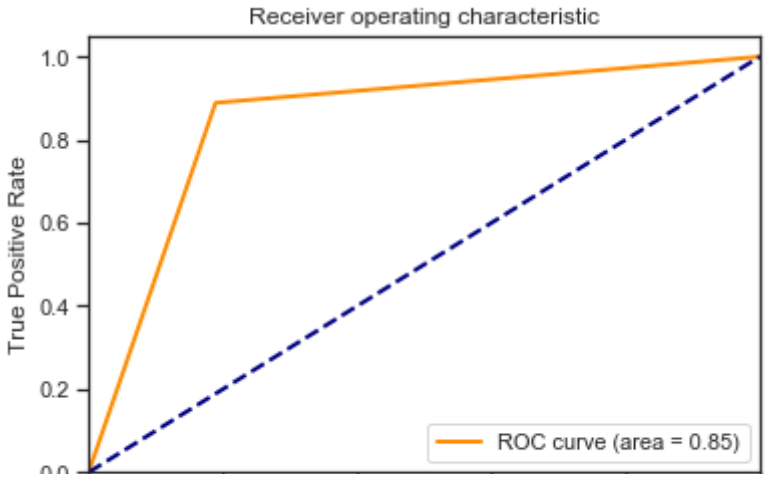
In [139]:
```
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

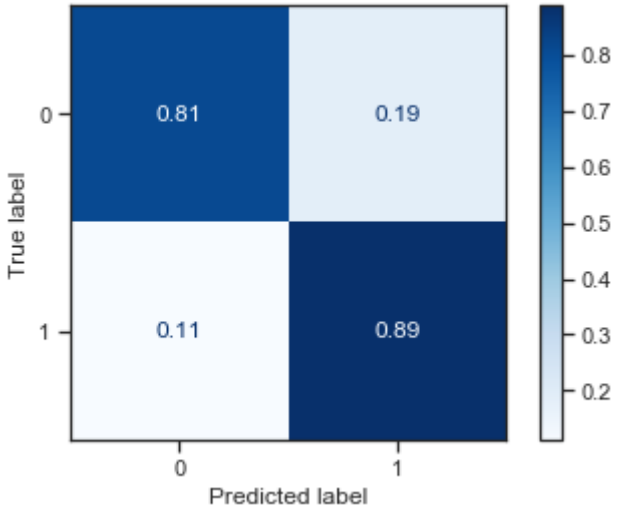
```
******************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tr
ue,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose
=0,
                   warm_start=False)
******************************************************
```



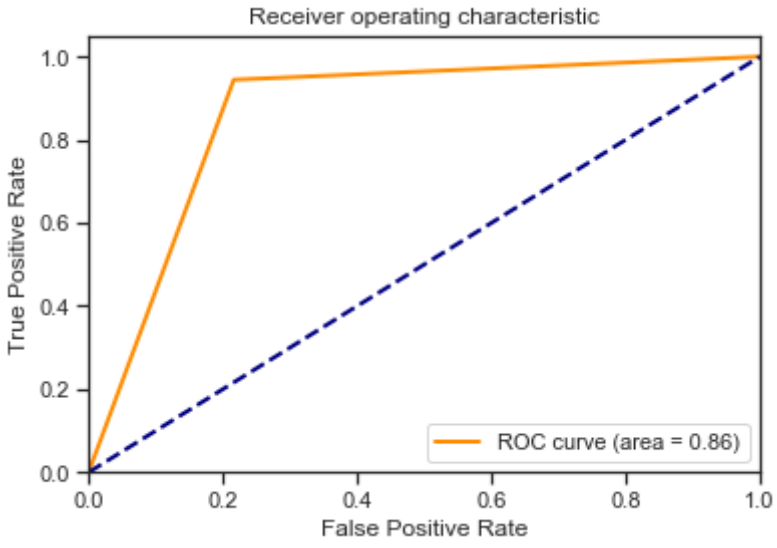


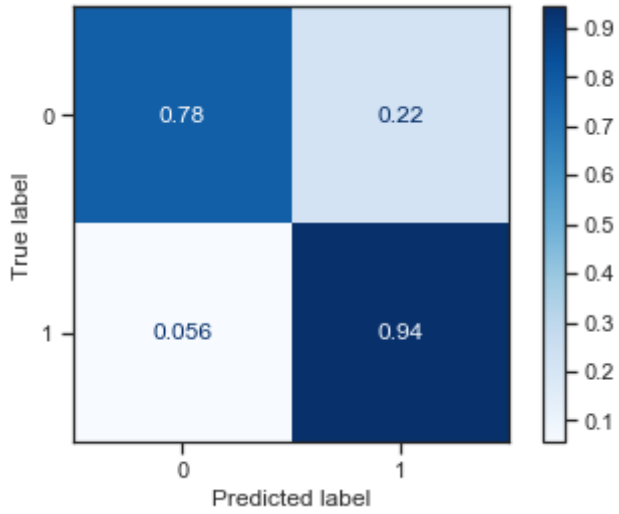

```
******************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
******************************************************
```



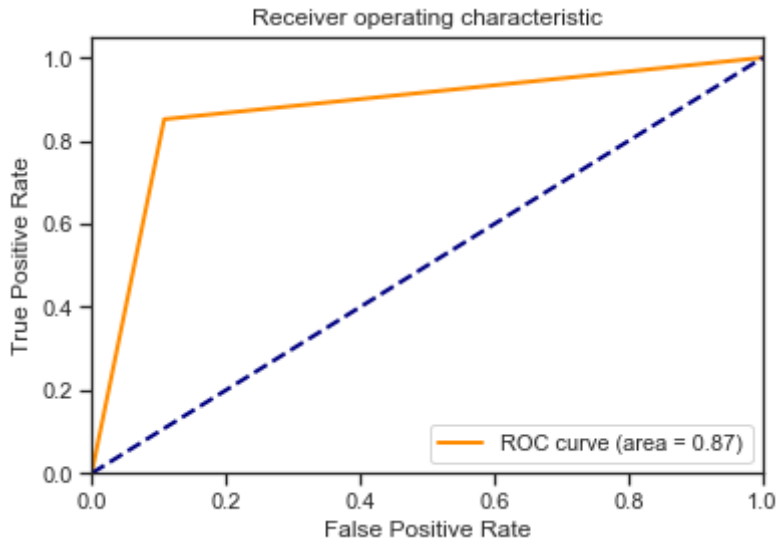


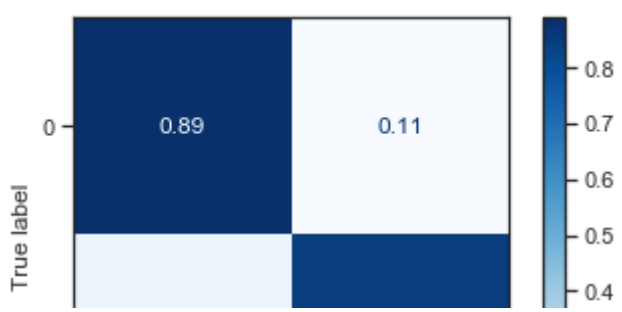

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.
0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
**************************************************
```



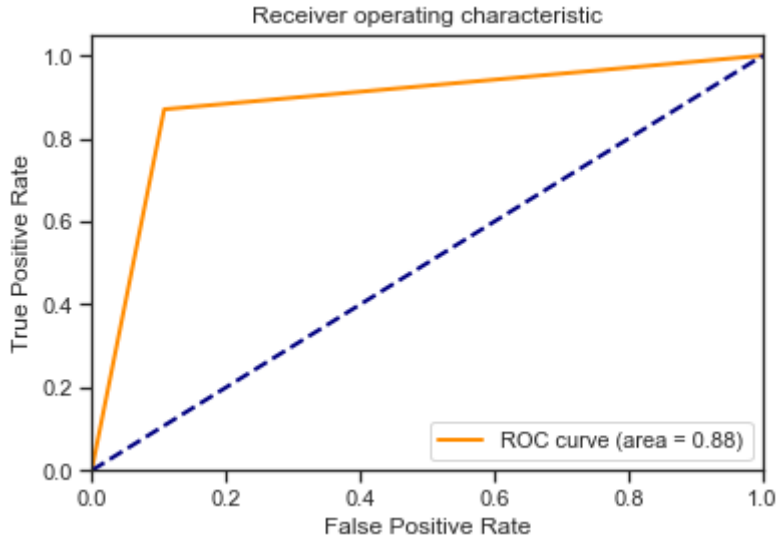


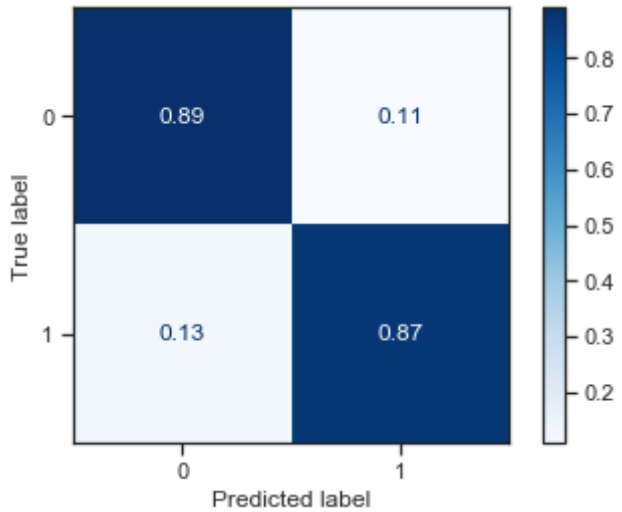

```
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gin
i',
                       max_depth=None, max_features=None, max_leaf_nodes=
None,
                       min_impurity_decrease=0.0, min_impurity_split=Non
e,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecate
d',
                       random_state=None, splitter='best')
**************************************************
```





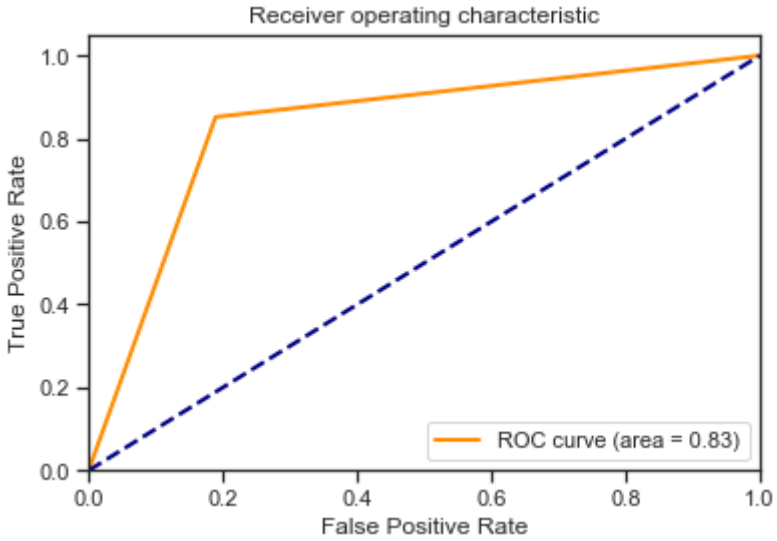

```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='au
to',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=Non
e,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
**************************************************
```

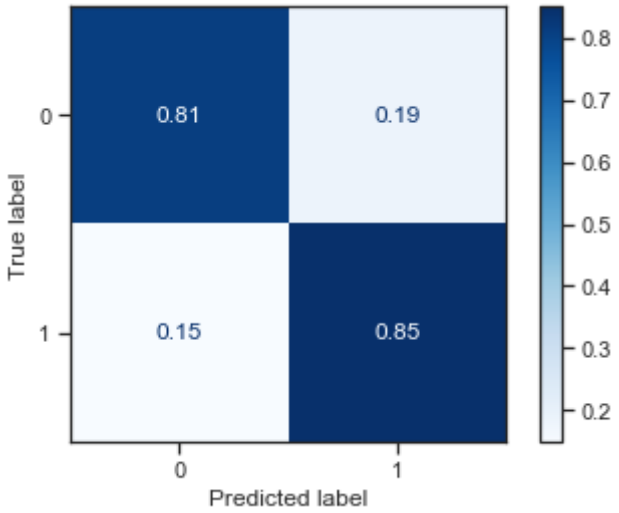





```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=
```

```
None,
                           learning_rate=0.1, loss='deviance', max_depth=
3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=
None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=10
0,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
******************************************************
```





8.2) Решение задачи регрессии

Определим модели:

In [140]:
```
regr_models = {'LR': LinearRegression(),
               'KNN_5':KNeighborsRegressor(n_neighbors=5),
               'SVR':SVR(),
               'Tree':DecisionTreeRegressor(),
               'RF':RandomForestRegressor(),
               'GB':GradientBoostingRegressor()}
```

Сохранение метрик:

In [141]:
```
regrMetricLogger = MetricLogger()
```

In [142]:
```
def regr_train_model(model_name, model, regrMetricLogger):
    model.fit(regr_X_train, regr_Y_train)
    Y_pred = model.predict(regr_X_test)

    mae = mean_absolute_error(regr_Y_test, Y_pred)
    mse = mean_squared_error(regr_Y_test, Y_pred)
    r2 = r2_score(regr_Y_test, Y_pred)

    regrMetricLogger.add('MAE', model_name, mae)
    regrMetricLogger.add('MSE', model_name, mse)
    regrMetricLogger.add('R2', model_name, r2)

    print('**************************************************')
    print(model)
    print()
    print('MAE={}, MSE={}, R2={}'.format(
        round(mae, 3), round(mse, 3), round(r2, 3)))
    print('**************************************************')
```

In [143]:
```
for model_name, model in regr_models.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
**************************************************
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=
False)

MAE=16.183, MSE=381.491, R2=0.251
**************************************************
**************************************************
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')

MAE=15.088, MSE=335.093, R2=0.342
**************************************************
**************************************************
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scal
e',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

MAE=15.612, MSE=371.069, R2=0.272
**************************************************
**************************************************
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')

MAE=17.23, MSE=467.871, R2=0.081
**************************************************
**************************************************
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes
=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)

MAE=16.006, MSE=393.287, R2=0.228
**************************************************
**************************************************
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_m
se',
                          init=None, learning_rate=0.1, loss='ls', max_de
pth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=N
one,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
```

```
                                validation_fraction=0.1, verbose=0, warm_start=
False)

MAE=15.644, MSE=373.526, R2=0.267
**************************************************
```

9) Подбор гиперпараметров для выбранных моделей.

9.1) Для задачи классификации

In [56]:
```
clas_X_train.shape
```

Out[56]:
```
(212, 7)
```

In [57]:
```
n_range = np.array(range(1,170,3))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[57]:
```
[{'n_neighbors': array([  1,   4,   7,  10,  13,  16,  19,  22,  25,  28,
31,  34,  37,
         40,  43,  46,  49,  52,  55,  58,  61,  64,  67,  70,  73,  76,
         79,  82,  85,  88,  91,  94,  97, 100, 103, 106, 109, 112, 115,
        118, 121, 124, 127, 130, 133, 136, 139, 142, 145, 148, 151, 154,
        157, 160, 163, 166, 169])}]
```

In [58]:
```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scori
clf_gs.fit(clas_X_train, clas_Y_train)
```

```
CPU times: user 3.71 s, sys: 117 ms, total: 3.83 s
Wall time: 1.6 s
```

Out[58]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=3
0,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=No
ne,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([  1,   4,   7,  10,  13,
16,  19,  22,  25,  28,  31,  34,  37,
        40,  43,  46,  49,  52,  55,  58,  61,  64,  67,  70,  73,  76,
        79,  82,  85,  88,  91,  94,  97, 100, 103, 106, 109, 112, 115,
       118, 121, 124, 127, 130, 133, 136, 139, 142, 145, 148, 151, 154,
       157, 160, 163, 166, 169])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring='roc_auc', verbose=0)
```

Лучшая модель:

In [59]:
```
clf_gs.best_estimator_
```

Out[59]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=25, p=
2,
                     weights='uniform')
```

Лучшее значение параметров:

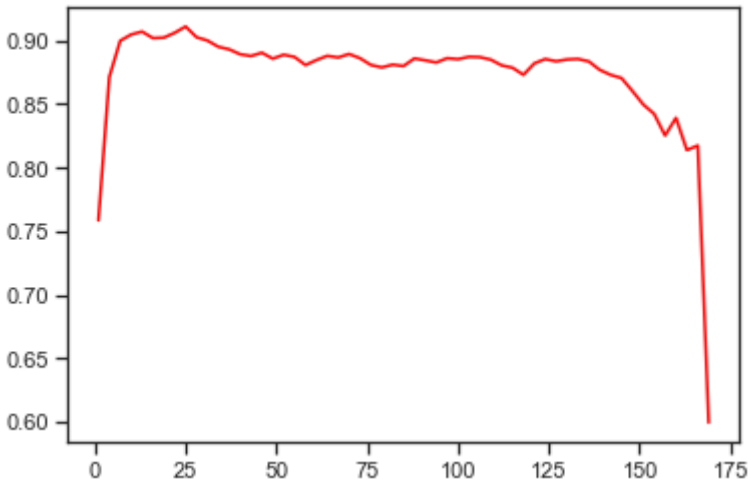
In [60]:
```
clf_gs.best_params_
```

Out[60]:
```
{'n_neighbors': 25}
```

Изменение качества на тестовой выборке в зависимости от К-соседей:

In [61]:
```
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'], color="red")
```

Out[61]:
```
[<matplotlib.lines.Line2D at 0x1a2dff6810>]
```



9.2) Для задачи регрессии

In [62]:
```
n_range = np.array(range(1,100,5))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[62]:
```
[{'n_neighbors': array([ 1,  6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 6
1, 66, 71, 76, 81,
        86, 91, 96])}]
```

In [63]:
```
%%time
regr_gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=5, scori
regr_gs.fit(regr_X_train, regr_Y_train)
```

```
CPU times: user 342 ms, sys: 2.47 ms, total: 344 ms
Wall time: 351 ms
```

Out[63]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsRegressor(algorithm='auto', leaf_size=3
0,
                                           metric='minkowski',
                                           metric_params=None, n_jobs=Non
e,
                                           n_neighbors=5, p=2,
                                           weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 1,  6, 11, 16, 21, 26, 3
1, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81,
       86, 91, 96])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring='neg_mean_squared_error', verbose=0)
```

Лучшая модель:

In [64]:
```
regr_gs.best_estimator_
```

Out[64]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=16, p=2,
                    weights='uniform')
```

Лучшее значение параметров:

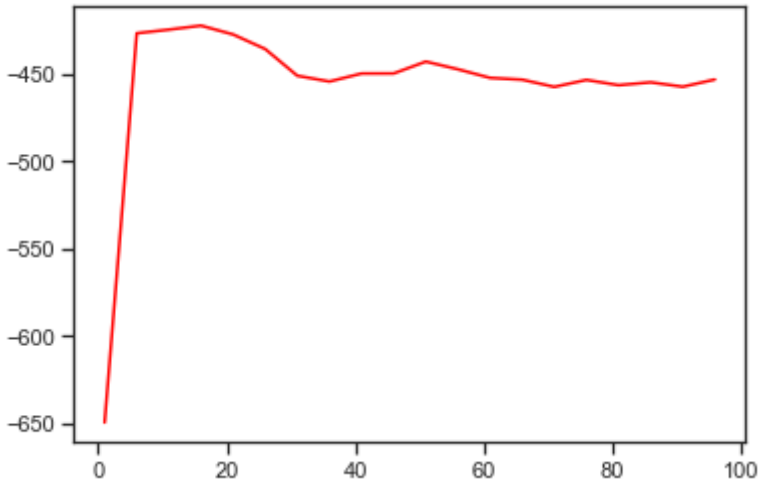
In [65]:
```
regr_gs.best_params_
```

Out[65]:
```
{'n_neighbors': 16}
```

Изменение качества на тестовой выборке в зависимости от K-соседей:

In [66]:
```
plt.plot(n_range, regr_gs.cv_results_['mean_test_score'], color="red")
```

Out[66]:
```
[<matplotlib.lines.Line2D at 0x1a2e105f90>]
```



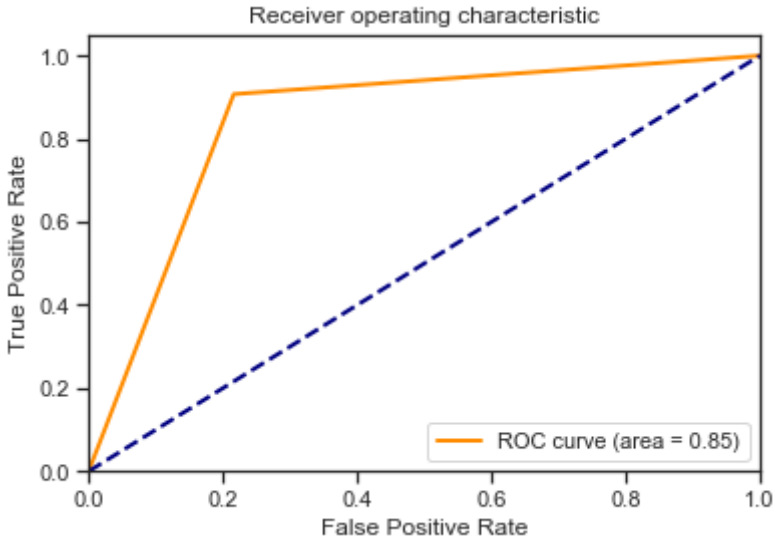
10) Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

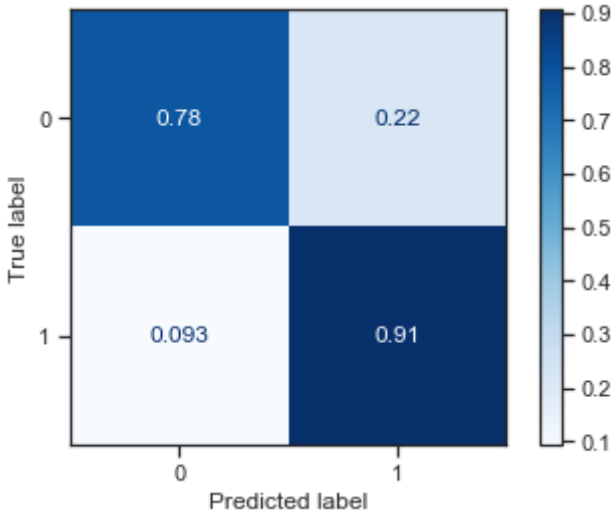
10.1) Решение задачи классификации

In [67]:
```
clas_models_grid = {'KNN_25':clf_gs.best_estimator_}
```

In [68]:
```
for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

```
******************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=25, p=
2,
                     weights='uniform')
******************************************************
```





10.2) Решение задачи регрессии

In [69]:
```
regr_models_grid = {'KNN_16':regr_gs.best_estimator_}
```

In [83]:
```
for model_name, model in regr_models_grid.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
**************************************************
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=16, p=2,
                    weights='uniform')

MAE=17.623, MSE=437.645, R2=0.141
**************************************************
```

11) Формирование выводов о качестве построенных моделей на основе выбранных метрик.

11.1) Решение задачи классификации

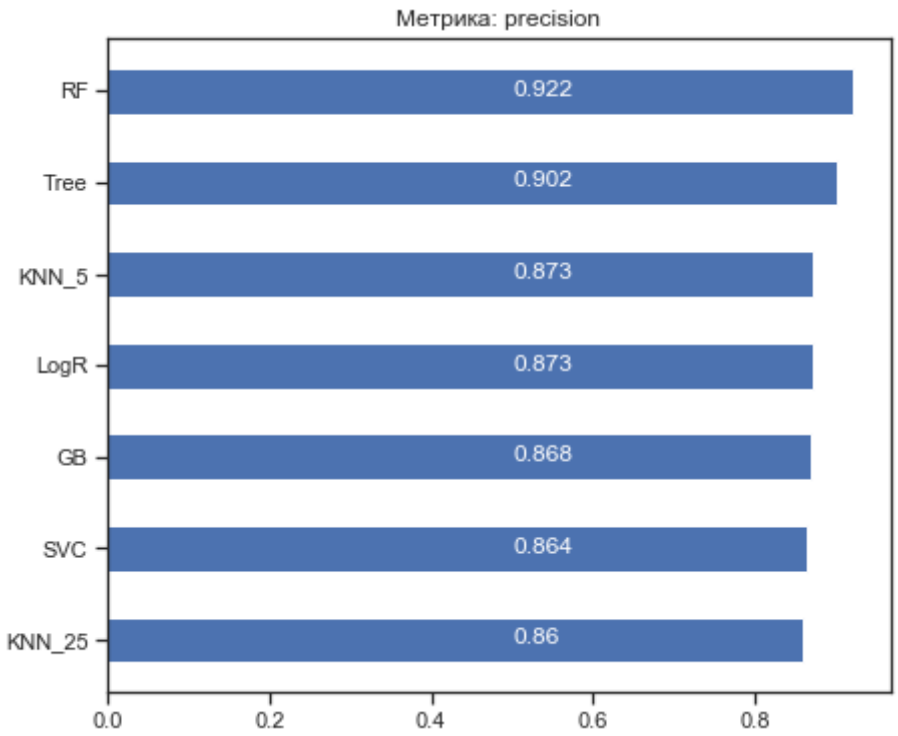
Метрики качества модели:

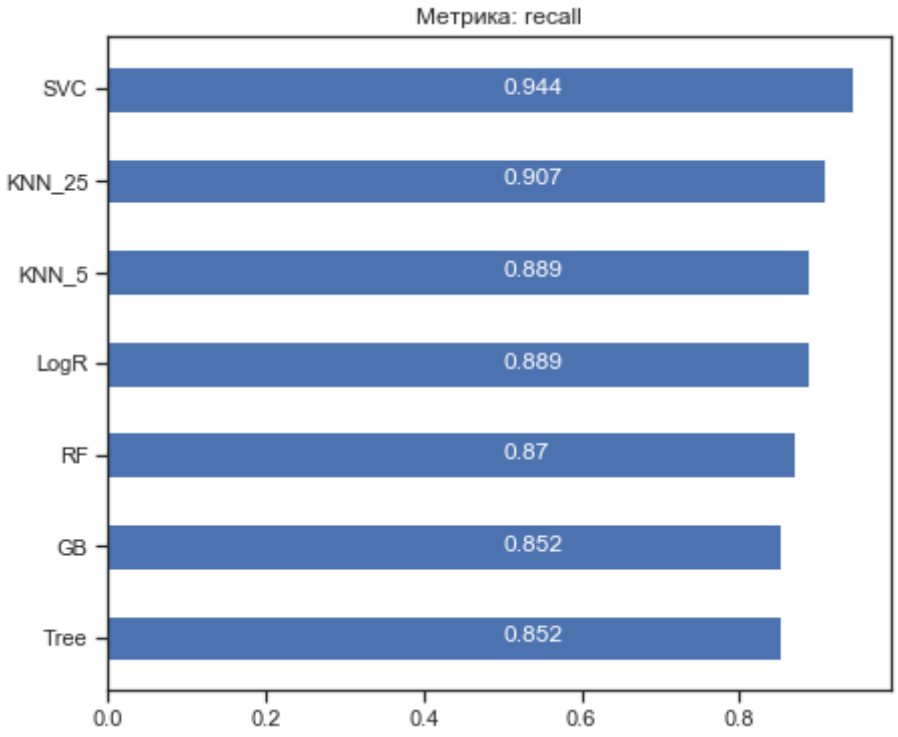
In [71]:
```
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

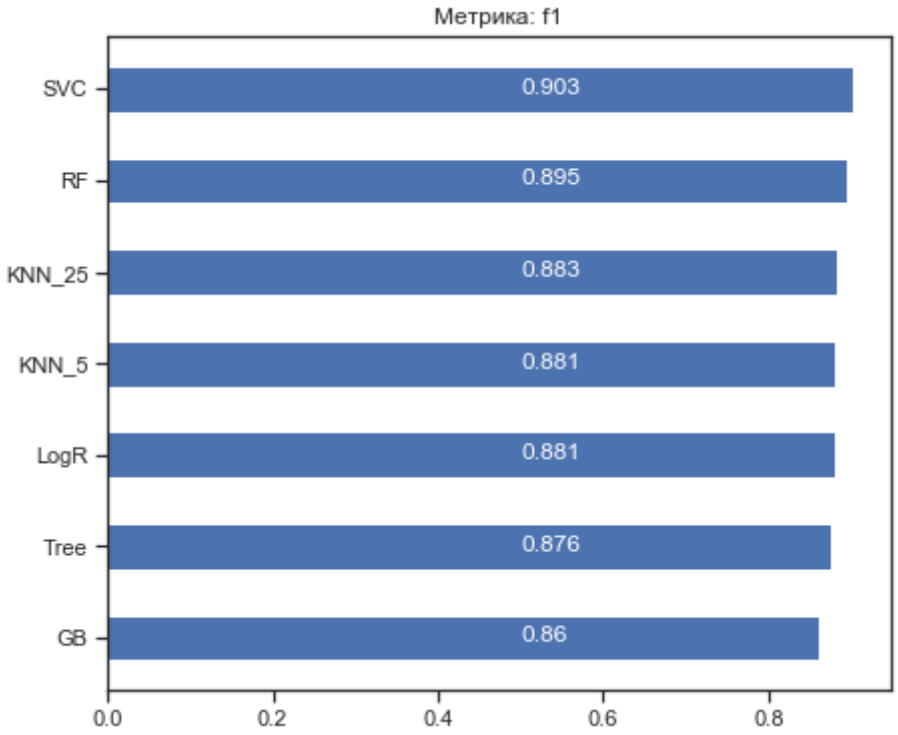
Out[71]:
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

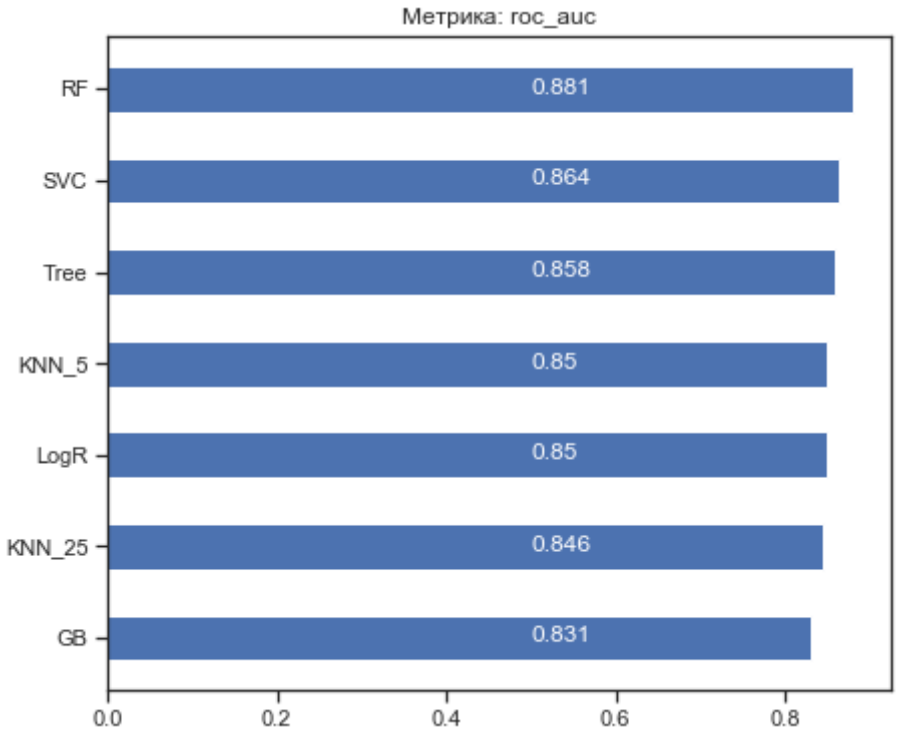
Графики метрик качества модели:

In [72]:
```
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```




Метрика: recall
SVC
KNN_25
KNN_5
LogR
RF
GB
Tree
0.944
0.907
0.889
0.889
0.87
0.852
0.852
0.0
0.2
0.4
0.6
0.8


Метрика: f1
SVC
RF
KNN_25
KNN_5
LogR
Tree
GB
0.903
0.895
0.883
0.881
0.881
0.876
0.86
0.0
0.2
0.4
0.6
0.8



Выводы: на основании четырех метрик лучшими оказались модель случайного леса и модель опорных векторов. Если подобрать хорошие гиперпараметры для этих моделей, можно сделать модель еще более точной.

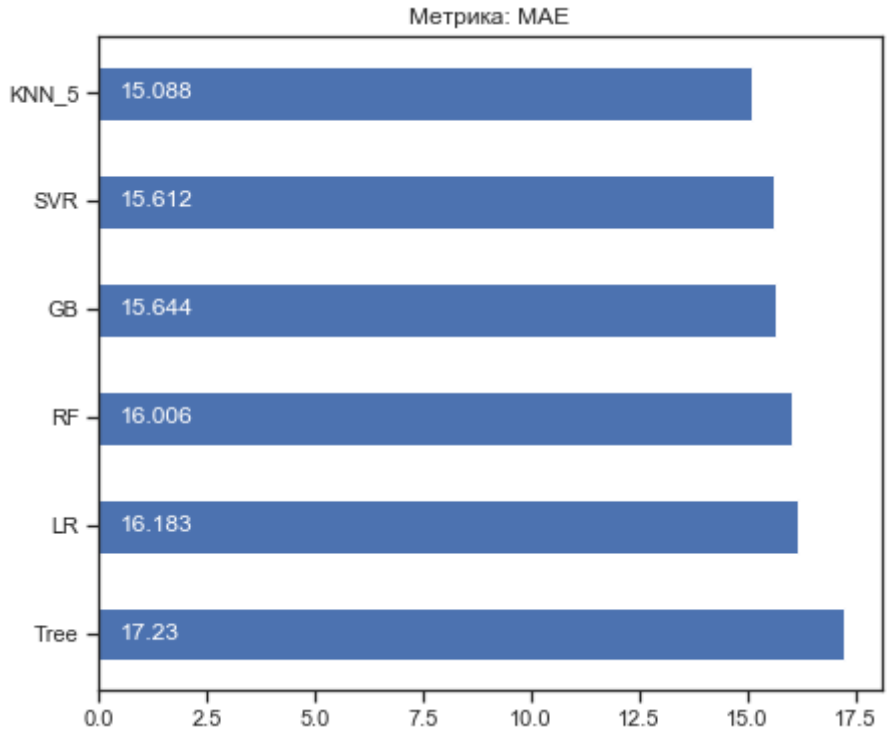
11.2) Решение задачи регрессии

Метрики качества модели:

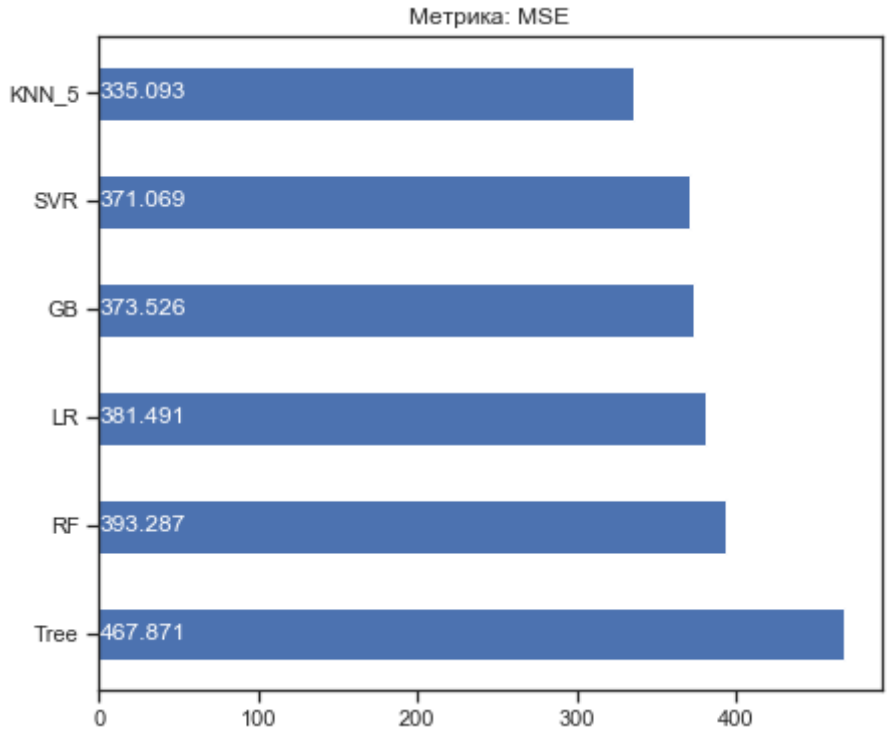
In [73]:
```
regr_metrics = regrMetricLogger.df['metric'].unique()
regr_metrics
```

Out[73]:
```
array(['MAE', 'MSE', 'R2'], dtype=object)
```

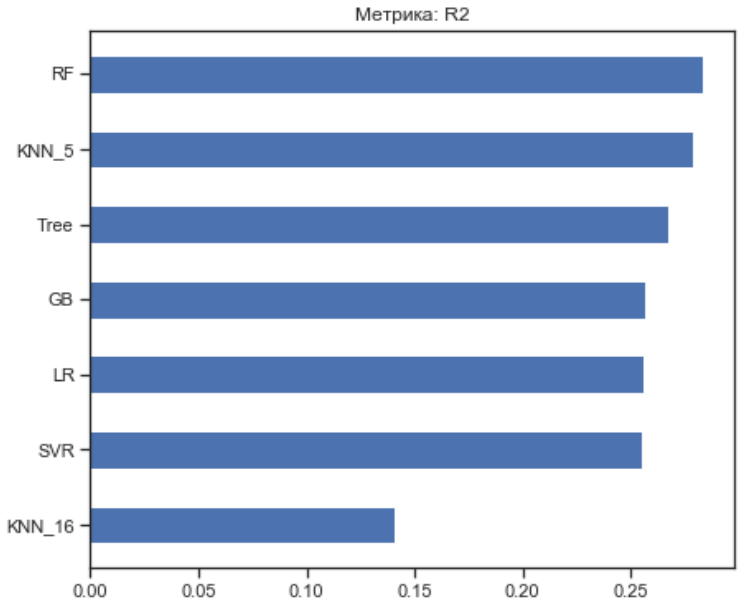
In [148]:
```
regrMetricLogger.plot('Метрика: ' + 'MAE', 'MAE', ascending=False, figsize=(
```



In [151]:
```
regrMetricLogger.plot('Метрика: ' + 'MSE', 'MSE', ascending=False, figsize=(
```



In [76]:
```
regrMetricLogger.plot('Метрика: ' + 'R2', 'R2', ascending=True, figsize=(7,
```



В общем случае можем сделать вывод, что методы опорных векторов и ближайших соседей показали себя лучшим образом. Гиперпараметры подобраны далеко не лучшим образом, хотя разумеется, есть куда стремиться